UNIVERSITY OF TURKU

Computer, Run Program

Assessing Viability of Audio Interfaces for Interactive Fiction

Department of Computing

Master's thesis

Author:
Liisa Peippo

March 2025

Master's thesis

Subject: Computer Science

Author: Liisa Peippo

Title: Computer, Run Program: Assessing Viability of Audio Interfaces for Interactive Fiction

Number of pages: 65 pages

Date: March 2025

Interactive fiction is a genre with a lot of potential and very enthusiastic communities. In this thesis, I investigated technological innovations in the genre, and then made an attempt on an experimental system of a fully voice-based interactive fiction game that is both controlled by speech and gives output by speech, without requiring a visual user interface. This did not wholly succeed, since the system provides some basic command-line output, but both the speech control and the speech output are functional, albeit not completely without problems.

For the purpose of building the system, I looked into different products that do the required functions for the features: speech recognition, speech generation, and narrative engines for interactive fiction. Then I tested players' experiences with the system, by recruiting volunteers, having them play the game and reporting their experiences with it.

The results remained inconclusive due to the small number of testers, but the data showed that further research might be in order, since none of the testers rejected the format outright. The results also gave ideas on how to improve the system further. The voice output is not ideal, and the speech recognition for the speech control could be more accurate. While the narrative engine serves its purpose for simple games, it is not suitable for more complex stories. For more conclusive research in the future, these problems should be addressed.

Key words: interactive fiction, speech interaction, audio interface, voice based.

Table of contents

1 Introduction

Interactive fiction is a computer game genre that emerged on the era before graphical user interfaces. When computer technology improved, and graphical games started to become first available and then more popular, commercial interactive fiction faded away. Despite this the genre did not disappear. Instead, it became a pursuit of hobbyists and enthusiasts, making games for the community for free. Interactive fiction evolved to encompass more kinds of products, including visual user interfaces and more graphical games, although the core of its gameplay is still text. While it is seen as a niche genre for most, even commercial interactive fiction has emerged again.

This thesis explores the technological evolution of interactive fiction, especially the less common forms of it. Continuing that theme, the work attempts to experiment on a fully audio-interfaced game and, using a proof-of-concept program, to test how players would react to such a concept. For that reason, the literature review in Chapter 2 focuses mostly on innovations about what interactive fiction *could* be, rather than research on what it *is*. Some of the latter is also included, in order to explain what the subject is, how users have experienced it, and what the players' expectations are.

The research question and its two secondary sub-questions for this thesis are the following:

- Would a fully voice-based interactive fiction format be considered interesting?
    - Does the player's previous experience with interactive fiction as a whole affect their opinion on this matter?
    - How do the player's first language and other background details affect the player experience with the proof-of-concept system, and interest in the format?

The proof-of-concept system built for testing these questions is a simple system that has a fully voice-based interface. It has a minimal text-based interface to show error reports and advice for pronunciation, but otherwise it is both controlled by speech and providing speech output. It has following parts:

- Text-to-speech part, that speaks out loud what would otherwise be included in a visual user interface, behaving like a screen reader for the program.

- Speech recognition part, that takes the user's spoken commands and translates them to text that can then be interpreted as commands.
- Engine part, that takes the command from speech recognition and decides the appropriate action based on it, and feeds text to the text-to-speech system to read out loud. It is both the functionality behind the user interface, and the system that controls which part of the game is showed next.

In an attempt to find answers to the research questions, the thesis consists of a look into technologies required to make voice-based interactive fiction possible, and an experiment where user responses for such a system was tested. The available technologies were researched by finding what kind of solutions are currently available for public use. The data on user responses was gathered by having testers play the experimental game and filling a questionnaire about it and the relevant background details, and observing testers play the game. This data was then used as a basis for discussion about the viability of this avenue.

Chapter 3 offers a deeper look on what interactive fiction currently is. It explains different types of interactive fiction games and systems made for creating them. It also explains what the important technologies behind audio interfaces are and gives examples of currently available systems that use some parts of them.

The experimental system is explained in Chapter 4, and Chapter 5 explains how the data about the user experience was gathered. In Chapter 6 the results are analysed, and finally in Chapter 7, the direction for future improvements of the experimental system and its functionality for future research on the subject are discussed.

2 Related works

This chapter provides a summary of the history of interactive environments and introduces some systems and frameworks that were created for interactive storytelling. A more thorough explanation of what interactive fiction is, and technical solutions for what this work could potentially use, can be found in Chapter 3.

2.1 History

Interactive fiction emerged during the 1980s. In 1989, Richard Ziegfeld [1] defines it as "literature delivered via software rather than print books". Furthermore, it is not necessarily only text, but may also include graphics and audio. The genre, or potential genre as Ziegfeld discusses in length on whether interactive fiction is one, has also a third significant component: reader involvement. There is a certain reader-author communication, where the reader makes choices, and the story reacts to them, in the ways the author has provided. This can lead to minor differences, like different dialogues, or completely different story branches, which can change the story's ending based on player's actions, or otherwise give alternate scenes. Individualization is also included in this: the reader may, for example, select the level of detail in descriptions in the story, or change the narrative perspective.

Andrew Stern [2] talks about interacting with characters in interactive fiction. The player usually assumes the role of a character, who then interacts with the environment and other characters, so it is especially important to understand how characters function. Since most stories are about people, the characters need to behave, talk, and be able to be interacted with, in ways that people do. However, it is not necessary for the characters to be life-like models of humans. What is necessary is that players *believe* they do so. Instead of internally "alive" characters, the story needs to provide an illusion of such. Stern describes this as "user-perception-based approach", using the virtual pet characters in PF Magic's Virtual Petz game series as an example. These characters are interactive cat and dog characters who act autonomously but can be interacted with and given commands to via a mouse. They have personalities and emotions, which they express the same way an actor does on stage, and they form relationships with the user and each other. Despite the characters not being alive, players form emotional relationships with them. People want to be engaged in a story, how the characters are cognitively modelled is not important. The Virtual Petz characters are animals, and as such, user interfaces for interacting with them do not have the same requirements that

interacting with humans do. Humans need to be able to speak, be understood when speaking, and be spoken back to. Humans also need to be able to communicate with voice, expressions, and body language. This requires a completely different user interface that cannot be formed simply by keyboard and mouse. It requires microphone and video camera, but it also requires the system in the background to be able to decipher all the nonverbal communication these technologies capture, in addition to the natural language processing required to successfully understand the unrestricted spoken words given by the player and text generation for the words spoken by the computer-controlled characters, to fully realize this experience. To create true interactive fiction, in its most interactive form, would also require the story to be able to be generated without unrestricted options for the player, something which would require the system to dynamically generate plot in the background.

Janet H. Murray [3] discusses forming coherent narratives in interactive fiction. A story is not just any sequence of events, but a sequence of events formed into a coherent plotline and shaped by the author's experience of the world. In a participatory medium, the interactor's actions must also be interesting. Without anything to connect emotionally, interactors lose interest. Since interactive experiences that offer engagement tend to require a high learning curve (or at least did in 1998, when the essay was published), ordinary readers perceive computer-based narratives as something incomprehensible. The word "nonlinear", which is used to describe interactive narratives that have branching plots, or multiple simultaneous plots, is understood as "nonsequential", as if interactive narratives would not follow any plot or have any causation. It is true that a story is shaped as much by what is left out as it is shaped by what is mentioned, but interactivity does not mean refusing to shape the plot. Design principles must answer to questions about what rules the world follows, how the signal the interaction points, how to communicate world boundaries and still offer the sense of agency, and how to entice the interactor into interacting. Experimental narratives can help authors discovering the limits and new conventions of the format and encourage them to practice the multisequential thinking required for authoring such stories.

Glorianna Davenport [4] describes a possible interactive environment, where real-life objects and entities would perform as interactive actors and components, by using windows and other displays that would act as screens mapping reality, and potentially giving different results based on user preferences and options. This could be used for purposes like providing entertainment for children on long car drives, modelling web searches as planting trees, or creating interactive pets who act and behave like real ones. The last one, a virtual dog called

Duncan, had already started being in development in 2000, when the article was released, although it was not yet as advanced as planned. Not all these scenarios would be easily achievable, if at all. Even if they are, it would not be certain if they would be marketable, but the possibilities are there.

James Pope [5] conducted an empirical study on readers' experiences on different works of interactive fiction. The study included seven works, all of which displayed a narrative with typical features of novels; the participants were all experienced computer users and readers. The results showed that although most of the participants were interested in reading more of interactive fiction, it was because the media was interesting, not because the experience was good. In fact, the experience itself was confusing for many of the participants, due to the conventions of the media being different from what the test subjects were used to, be that a print or a gaming functionality. The study suggests that to be "successful", interactive fiction needs to build a consistent and complete story between all the elements, and that the interface needs to be usable, but giving the player a cognitive overload with too many options and elements needs to be avoided.

Diego Gonzales and Andrew S. Gordon [6] conducted an experiment to compare the differences in player interactions between text-based and speech recognition interfaces in interactive narrative. The experiment used the same narrative structure and plot for both versions, apart from the fact that since the original version was created as an audio drama and not an audio *book*, the text-based version had to include some additional text to describe events and information which were provided by sound effects and the like in the audio version. The options for players were not simple choices, but instead allowed the player to phrase their commands the way they wanted, which relies heavily on algorithms and accuracy in automatic speech recognition, as well as the authors' ability to predict the players' actions. The results show systematic differences in how the players format their responses between input types – players had a higher tendency to narrate their actions in text input format, although the player responses in total were of widely varying types – but despite this, no difference was found in player experiences described in post-experiment questionnaires.

Sergio Nesteriuk [7] discusses how videogame accessibility is important in the modern world, where gaming is a part of life for a large part of population and multiplayer games provide an essential way to communicate with other people. For blind and visually impaired people (BVIP), this means games that can be played without depending on visuals. This does not

necessarily mean that the games could not have graphics: well-functioning audio cues can provide the same information the graphics do, and they can do that at the same time, as long as the design of the game supports such functionality. This can help people with all levels of vision to enjoy the game, which is important for inclusion. Paradoxically, finding accessible videogames was easier among the early commercial ones. Due to the technological limitations at the time, many of the games with elaborate narrative plots relied heavily on written text, which could then be played as audio by using assistive technologies like screen readers. As technology developed, text gave way to graphics and complexity in gameplay and interfaces. The same technological development gives opportunities for inclusive games, such as audio games. Even so, audio-based games have not reached much mass appeal: they are mainly created by academic projects, enthusiasts, and small indie developers, not AAA studios. The gameplay is also rarely designed to be fun for BVIP, focusing more on loss of sight than empowering of hearing.

Laura Okkema [8] seeks to direct interest in game studies to smart speaker systems. These systems consist of voice-interactive sets of speakers controlled by an artificial intelligence, which allows creating voice-controlled audio games. Voice interfaces became available after technological developments made speech recognition, microphones, and speakers accessible to players. First profitable platform to offer such features was Nintendo's Famicom, on which, once developers started experimenting with it, a wide variety of voice-based inputs was offered in various games. Overall, early trends in voice input focused on relationships in Japanese games, and command-and-control in Western games; however, voice is usually only one of possible input methods. Unlike these, smart speaker systems have voice-only interfaces: they are, essentially, always listening. This brings with it ethical and privacy concerns, since the big companies' speech recognition systems, once activated, upload everything they hear into servers for processing and interpretation, where they keep it stored and used to unknown ends.

2.2 Implementations

Röber et al. [9] introduces a system for creating interactive audiobooks, which mix game elements with regular audiobooks. Based on their prior experiences, they decided that user movements would have to be restricted, since free exploration leaves players getting lost: this requires control from the story engine to hold the narration. Since these stories are non-linear, the engine uses *story graphs*, flowchart-like structures that contain story pieces and

minigames, allowing the user to control the story based on the boundaries the author has decided. The system allows jumping from one point to another and using internal logic based on player's choices; this requires author segmenting the story in suitable chunks in proper arrangement in the graph. The system can also act without user input, in which case it chooses automatically or, as the system calls it, *auto-chooses* options based on what the player has selected thus far. This mode can be entered by deciding non-action when the game asks for input, but it can be changed back to interactive mode at any time. The interface is using a gamepad control, since although speech recognition would be most convenient option, it was not available enough at the time of creation of the system. The framework itself consists of authoring and runtime parts, which allow a creation and presenting a story: it is written in Java, Delphi, and C++, and handles the audio with OpenAL.

Nick Montfort [10] introduces Curveship, an interactive fiction system that uses natural language generation for varied narration. This is important for the experience, because a story is not only the events that take place in it, but also the way they are narrated. A timeline does not equal a great work of literature, even if the events are the same. IF systems have their differences, but they are similar in offering "parser", which transfers the user-given commands to actions, and the rest of the program, which handles everything else. Curveship also has a parser (called "Recognizer"), but its structure is more modular, each module handling a different function the system needs, including "Clarifier" that clarifies unclear input, and "Joker" that deals with saving, loading, and other operations like such. Most notably, it has separate "Simulator" and "Narrator" modules. The Simulator updates the world states of the story, following the choices the player makes and their consequences, and it can be independent of the human languages used. The Narrator, on the other hand, is very dependent on the human languages used, as it is the module building the narration based on the world models and the plan the author has for the story. All these models work together to build the discourse that takes place in the story, the Recognizer taking account the user input, the Narrator producing the system responses, and the Simulator working on the background to update the world models the Narrator draws information from. The events of the story do not need to take place in the chronological order, or not in the same order in every time the story is played. Curveship includes a tree presenting the structure of the events. The story is based on this tree, the plan for the narration, and the structure of replies as it is played.

Da Hyeon Choi [11] suggests technical solutions for interactive dialog-based storytelling. The system is called LYRA, Learning Youth Reading Assistant, and it is directed to helping

childhood development by providing storytelling experiences for children whose parents are too busy to read to them. It is using dialogue generation, speech recognition, and speech synthesis, to create similar communication that children have when their parents read to them, giving the listener the possibility to ask questions and have them answered. At the time of when the article was written (it was published in 2019), the systems were still prototypical due to technological limitations: the level of natural language processing technologies the system requires to function correctly were lacking in available systems like Apple Siri or Amazon Alexa. The idea of interactivity in the system is not limited to asking questions: it is meant to provide the listener the ability to choose the fate of the characters in the story, thus providing multiple possible paths for the story. The prototype uses a conversational model with limited capability to answer questions, and thus the study included sample questions for the users to ask. However, the system was capable of handling limited cases of questions different from its training data.

Usnea is a tool by Ben Swanson and Boris Smus [12] for authoring interactive fiction. It provides both a story editor with a visualized graph for the story nodes, which manages the parsing of the story and how it changes reacting to user input, and a language model with semantic parsing for dialogue management, which controls the story to be parsed. For the language model, the editor also provides an interactive tester for defining utterances that should or should not apply to a specific story node. For more complex storytelling purposes, the nodes are not only direct responses to prompts given by the player, but also an underlying global state of the gameworld, which may affect the nodes and the conditions by which they apply. The tool is implemented using the JavaScript library Angular, uses a Firebase cloud database, and is an open-source solution. This allows custom data to be injected, as well as provides modularity and customization in the tool itself. This is meant to be as a flexible model and authorship tool, which can be upgraded by a better and more fine-tuned models, all the while allowing the use of a semantic parsing system by an author with no knowledge of how any of this works.

Matthew Hausknecht et al [13] introduces Jericho, an open-source Python environment for supporting machine learning algorithms to connect with interactive fiction. It is built to work with parser-based games and supports a set of them. While it can be used with unsupported parser games as well, the functionality it provides with them is limited. Choice-based games, which are the other type of interactive fiction, are not supported. The system generates actions based on templates, which are game-specific. These are built by decompiling the game to find

the possible subroutines and vocabulary. It then combines the template with the vocabulary to produce the possible actions. For supported games, the system detects the game score, move counts, and changes in the gameworld. The world changes are detected by changes in world's objects, so it may not detect some changes that only change global variables, thus interpreting those actions as not valid. In practice, however, this is rare. Jericho was then evaluated by comparing its results with the results of four other, differently functioning interactive fiction algorithms playing the same games Jericho supports. The results show that simply taking random actions is not sufficient to solve these kinds of games, although more training for an agent with a game improves its score. More research is needed before a computer is capable of completing more complex parser-based games. Especially better template-based agents are needed, as well as making sure the agents can play games they were not trained on.

Amal S. Fadak and Mohamed O. Khozium [14] explores designs related to dialogue generation, a new technique for controlling interactive narratives. This is called Digital Interactive Storytelling (DIS). According to this research, digital interactive storytelling systems all contain a drama manager, which controls the story's mechanics and actions in the virtual world, and characters, who act and communicate to achieve their goals. Following these, the system generates sequences of organized and coordinated actions that construct a story. Story production is important for players to feel immersed and involved. The story can change based on player choice, or it can consist of multiple parallel storylines the player can participate in. Dialogue generation works within a context of the storytelling system, which can allow the player to direct the story without restricting what they can say. Such a system needs to be able to predict the player's speech and to generate content that completes the player's requests while being well-structured and following the discussion. This can be accomplished by Natural Language Processing (NLP) systems, using the subtypes Natural Language Understanding (NLU) and Natural Language Generation (NLG). Artificial intelligence systems can be used for story planning, allowing the player to take random actions and still maintaining the cohesion of the story.

3 Technical background

The work in this thesis focuses on interactive fiction and uses several existing technologies. This chapter explains what interactive fiction is in the context of this work, what kind of systems are currently available for building interactive fiction narratives, what kind of technologies the experimental features that were tested use, and what kind of systems for those features that are not used in the thesis are available. More information about the solutions used in the experimental system itself can be found in Chapter 4.

3.1 Interactive fiction

Interactive fiction is a computer game genre that has different definitions based on who is giving them, but in this work the term is used in the meaning defined by the Interactive Fiction Technology Foundation (IFTF). It is an organization focusing on archiving, distributing and developing interactive fiction as an art form [15].

> The IFTF defines interactive fiction–IF, for short–as a kind of video game where the player's interactions primarily involve text. Under this broad definition, we can find decades of IF work taking many interesting and innovative forms. [16]

Figure 1. The web version of Adventure has an interface with a picture of an old computer in addition to the actual game screen. The keyboard is not functional. [17]

The idea of interactive fiction can be found as far as 1941, where a novel with multiple endings was described in a story, although that was not yet actual interactive fiction. In 1960, an experimental short story with branching narrative was written. As an actual medium, interactive fiction appeared in 1975, when the first text adventure[1], named Adventure, also known as Colossal Cave Adventure (Figure 1), was written. In 1980s, these games became a

[1] Or perhaps second. While most sources, including the Interactive Fiction Technology Foundation itself, claim that Adventure is the original text adventure, a less-known game called Wander was written on the previous year, in 1974 [120].

major industry, but when graphical games became more popular (and more readily available), interest in text adventures diminished. Internet and the ease of digital game distribution with it, as well as readily available tools for making games, has made interactive fiction attractive to hobbyist and indie authors instead. [16]

Interactive fictions games are mostly text, and as such, require a lot of reading, which is why it is not always considered a proper videogame genre by gamers. The most well-known forms of interactive fiction are *parser-based* and *choice-based* [16], both of which are *text*-based. These games can include graphics, like a novel can include illustrations, but the gameplay is all done in text. Some people include also graphical games that otherwise follow the same format, such as visual novels, which are essentially choice-based games with graphics, but the interactive fiction community does not usually count them among true interactive fiction [18]. However, the definition itself supports calling also graphical games as interactive fiction, as long as the player's interactions primarily involve text. In Disco Elysium (Figure 2), the player character moves around in an isometric 3D world, but the gameplay consists mostly of clicking interaction icons that trigger text screens.

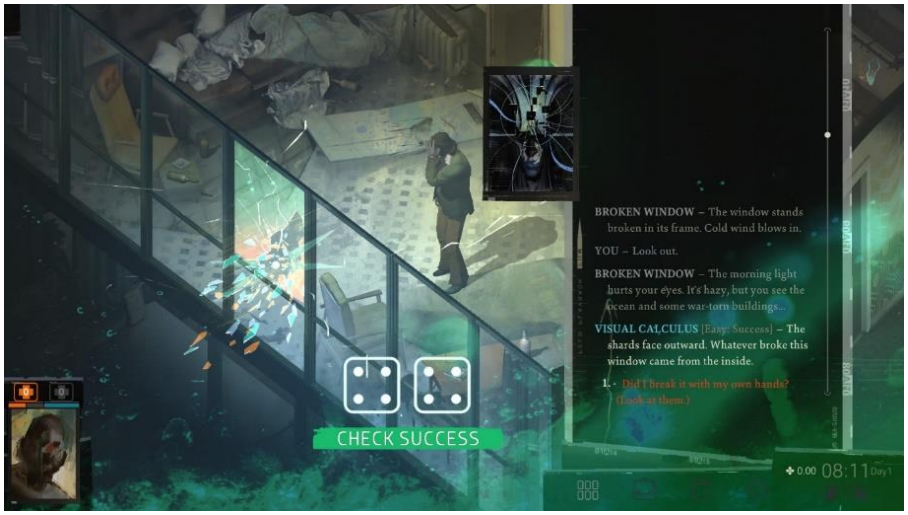

Figure 2. In Disco Elysium, almost everything happens in a text screen. This includes narration, making choices, and rolling dice for skill checks. [19]

Parser-based games, also known as parser games and text adventures, are the original form of interactive fiction [16]. These games started to appear in the 1970s, long before the genre itself was defined [20, 1]. Since there were no graphical user interfaces yet, these games were fully text, utilizing the command-line interface of the computers, first on mainframe systems and then later on personal computers, and finally internet and mobile devices [21, 22, 23].

As is typical for command-line programs, parser games are controlled by typing commands (Figure 3). The game then interprets those commands as actions, which are carried out by the main character. As a response, the game then generates the appropriate new state of the gameworld and its possible characters from the background system controlling the game. Different parser games and systems can have different vocabularies and understand different sentence structures. Some parsers use more limited vocabulary, while some others can understand more complex commands. Additionally, not every command is usable with every item or object, or at least using it may not be sensible; how the game acts when the player tries something unactionable differs from game to game and action to action. [16, 20, 23, 24, 25, 26, 27, 28]

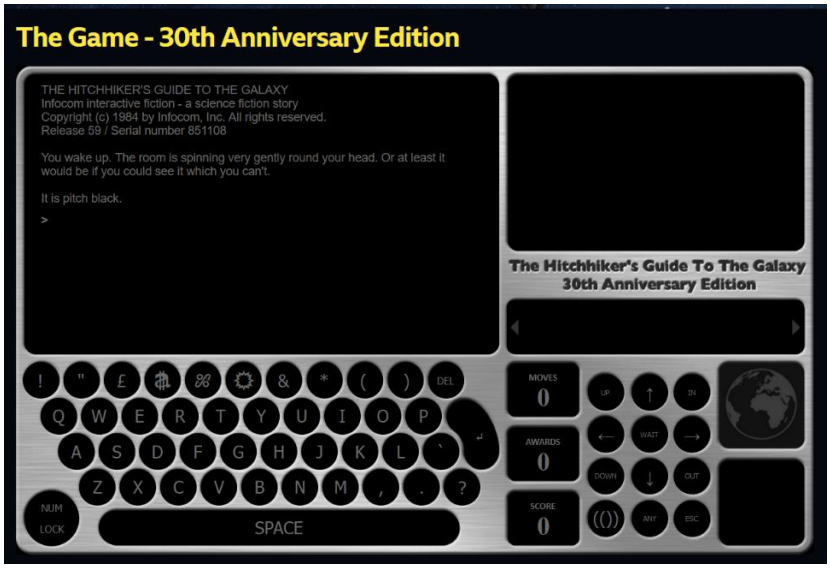

Figure 3. In the 30th Anniversary Edition of The Hitchhiker's Guide To The Galaxy, the interface resembles a full device. The on-screen keyboard is functional and can be used for input, but the actual keyboard of the device the page is viewed with works for input device as well and is arguably easier to type in. On mobile devices, the full interface does not appear at all on smaller screens, instead showing only the text window and forcing the use of the device's own keyboard. [29]

Parser games usually include puzzles for the player to solve, and some definitions call this an essential part of a parser game [24]. Some authors have also made parser games without puzzles, and these, too, have been well received by the players who have tried them, but they have stayed in the minority [30].

In choice-based games, there is no typing of commands. Instead, the game provides the player options to choose from, either by typing the number of the option with keyboard, if numbers are provided, or using mouse or an equivalent pointing system to pick the option from the provided list in a graphical interface [31, 32, 33]. Both methods can be in use at the same time (Figure 4). Choice-based games, more shortly choice games, sometimes also called as choose-

your-own-adventure games after a popular book series, function essentially like interactive novels, where the player can select different options about where the story goes [31]. These games can be either digital or physical, the aforementioned book series being an example of the latter [16]. Unlike printed books, digital games have the benefit of being able to use variables and having the capacity for longer stories than the physical ones, which soon become unwieldy the more the length and branching increases [33].

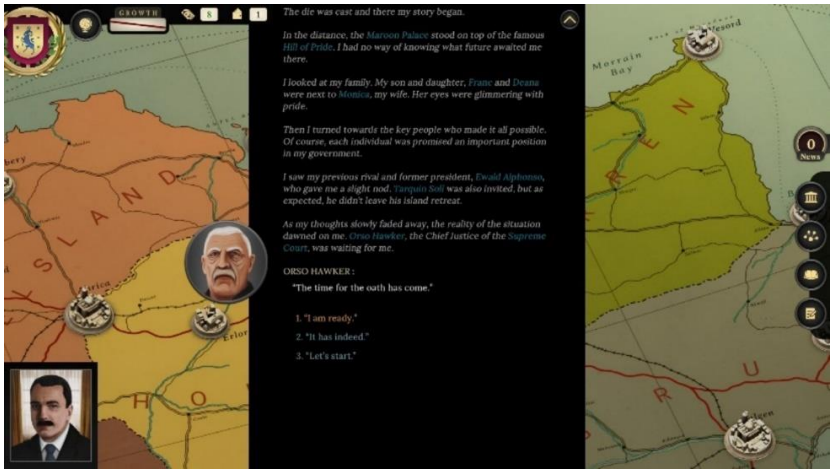

Figure 4. In Suzerain, the map is used to move from one text segment to another. The provided choice options can be selected both by clicking them and pressing the option's number on keyboard. [34]

Not all interactive fiction games can be easily categorized as parser or choice games. It is possible for a game to have features of both. These are called parser-hybrid games, or more shortly, hybrid games. However, since 2010s most parser games have used choice-based conversation systems (Figure 5) instead of the older typing-based ones. While these games are also using choice mechanics, they are not generally categorized as hybrids as long as those mechanics are not used outside conversations. [35]

Figure 5. Taco Fiction provides a numbered list of options the player can select in a conversation. Typing the option's number selects it. [36] Since the options include also actions the player can take and not just words they can say, the game is considered a hybrid [35].

As stated in the Interactive Fiction Wiki [35], hybrid games can take many forms:

- Parser games with choice sections, and vice versa. These games are mainly using one format but have sections that use the opposite one (Figure 6).
- Choice games with parser-like world models. These games are controlled by choices and made by traditional choice systems, but they otherwise behave like parser games, using similar structures and mechanics.
- Reduced parser games. These games are parser games, but their vocabulary is very limited to the degree of being almost choice-like.
- Parserless parser games. These games would otherwise be parser games, but they use choice-style buttons or links for input. Unlike the choice games with parser-like world models, these games use systems made specifically for parserless parser games.

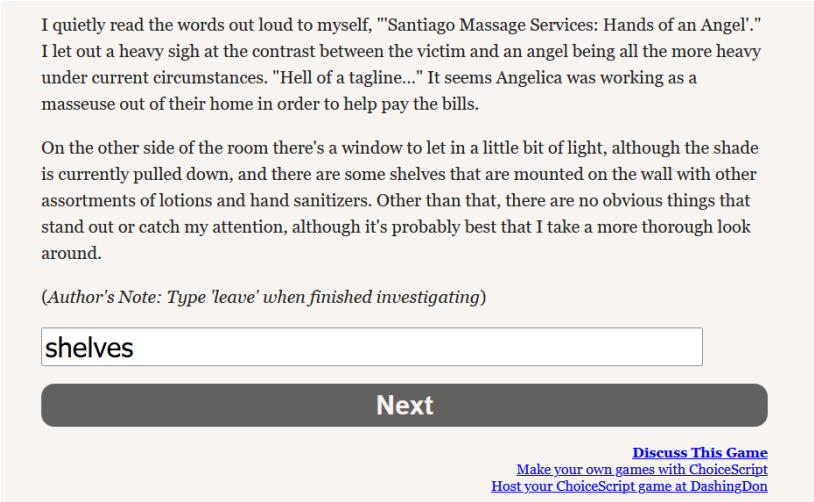

Figure 6. The Bureau is a choice-based game that has investigation scenes where the player types keywords picked from the scene to a text box to investigate them. [37]

Graphical games with interactivity in their content that otherwise do not fall under the definition of interactive fiction, usually employ choice mechanics and not parser ones for their interactive parts. Mostly this happens in conversations, where the player character has multiple options for what to say. This is so common that it is less a notable choice-based mechanic and more a roleplaying mechanic: there are, after all, only so many ways a conversation mechanic can be constructed. In addition to choices, the player character's actions in the gameworld, outside the conversations, can sometimes also affect the plot or other characters' reactions if the player character has, for example, robbed a specific container or has been caught lockpicking a door. This could, perhaps, be seen as a 3D version of a

parser-type model of the gameworld. While graphical games that are not interactive fiction, but which nonetheless use actual parser mechanics, are rarer, they also exist (Figure 7).

Figure 7. In Wasteland 2 (top image), conversation options are selected by typing keywords in a text box. Some options are provided, and they can be clicked, but that simply auto-fills the keyword to the text box. [38] In Her Story (bottom image), the gameplay consists of a search interface, where the player needs to watch video clips, pick keywords from them, and do new searches with the new keywords to find more video clips. [39]

3.2 Game engines

Interactive fiction engines are called *authoring systems* [40], but they fill the same role a game engine for a graphical game does. They provide the technology required for the game to function as a game, as well as provide support for the game mechanics.

3.2.1 Parser engines

ADRIFT, short for Adventure Development & Runner – Interactive Fiction Toolkit, is a parser-game development system by Campbell Wild. It is exceptional in being designed for authors who do not want to do programming, as opposed to systems designed for

programmers. Providing a GUI application, ADRIFT allows building a text adventure using visual elements. It has a graphical map system where items and characters can be placed with a few mouse clicks, and while earlier versions sacrificed functionality for usability, the newest versions of ADRIFT 5 are powerful enough to do everything a more programming-heavy authoring system does. ADRIFT games require a special runner, which can be used on a website (Figure 8), downloaded separately, or be compiled into the game download. [41, 42]

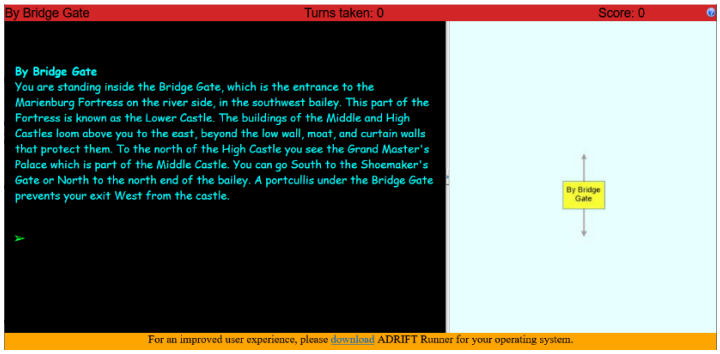

Figure 8. The Fortress of Fear, as viewed in ADRIFT web runner. On the right-side panel of the window, the game shows which room the player is currently in. [43]

Adventuron is a parser game engine with experimental but not much used support for choice games, created by Chris Ainsley. It provides a cross-platform editor that runs in a web browser, that also compiles the completed game into a single HTML file. It has a good parser, multimedia support, embedded markup language, and support for building 8-bit and 16-bit like games. It also supports custom fonts and has no upper limit for multimedia size, so Adventuron is not limited to retro-style games. It is available only as an education version for non-commercial use, although it nonetheless has full functionality. [44, 45] The use of illustrations (Figure 9) sets these games apart from most traditional parser games.

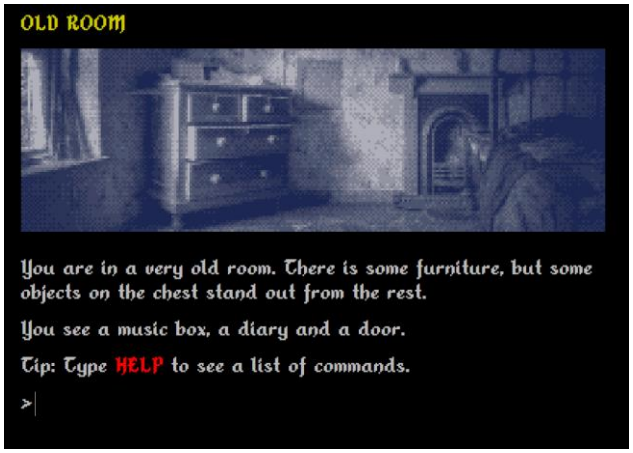

Figure 9. The Mansion as an example of an Adventuron game, showing an illustration on top of the page, a specific font, and parser functionality. [46]

Dialog (Figure 10) is an authoring system for parser games by Linus Åkesson under 2-clause BSD license. Although it also includes a choice mode, the choices are selected by inputting numbers the same way parser commands are inputted. Compiled Dialog games require an interpreter, but the system is capable of compiling into more than one format. [47, 48]

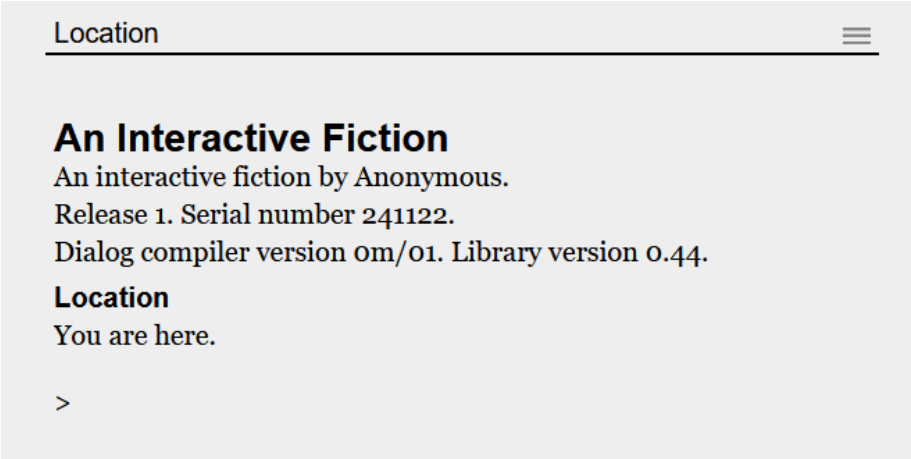

Figure 10. An example of how a Dialog game in a browser looks like, built using the sample data available on the online editor for the system. [49, 50]

Inform is one of the most popular interactive fiction authoring systems in the world, created by Graham Nelson. It has multiple variants and is licensed under Artistic License 2.0. It uses a subset of natural language, and its system is notably based on rules instead of objects. An editor is available for both a downloadable version and an online one (Figure 11). The compiled games need an interpreter, of which multiple versions are available for multiple operating systems. [51, 52, 53, 54]

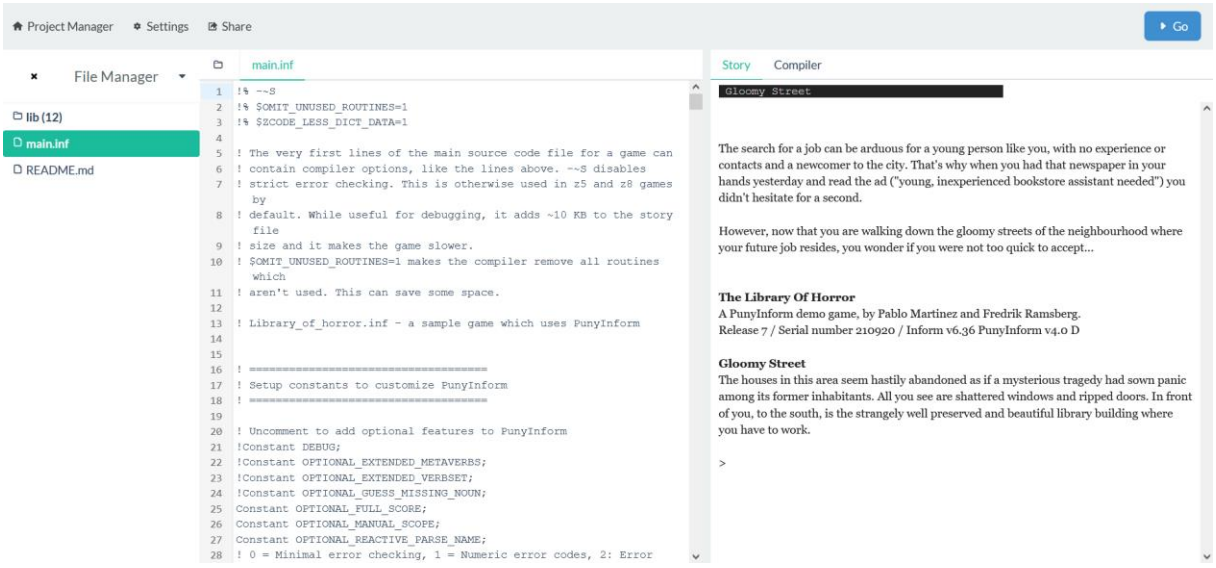

Figure 11. The online editor for Inform 6 with Library of Horror, a sample game that is provided in the online editor. The game can be seen running on the right-side panel of the window. [55, 50]

TADS, short for The Adventure Development System, is a parser authoring system by Michael J. Roberts. It is free software with its source code also available. TADS uses object-oriented programming, and creating a game using it requires actual programming. Unlike most popular authoring systems, it supports online play and multi-user games. TADS can compile games as stand-alone executables, games requiring interpreter, and games running on a web browser (Figure 12) [56, 57]

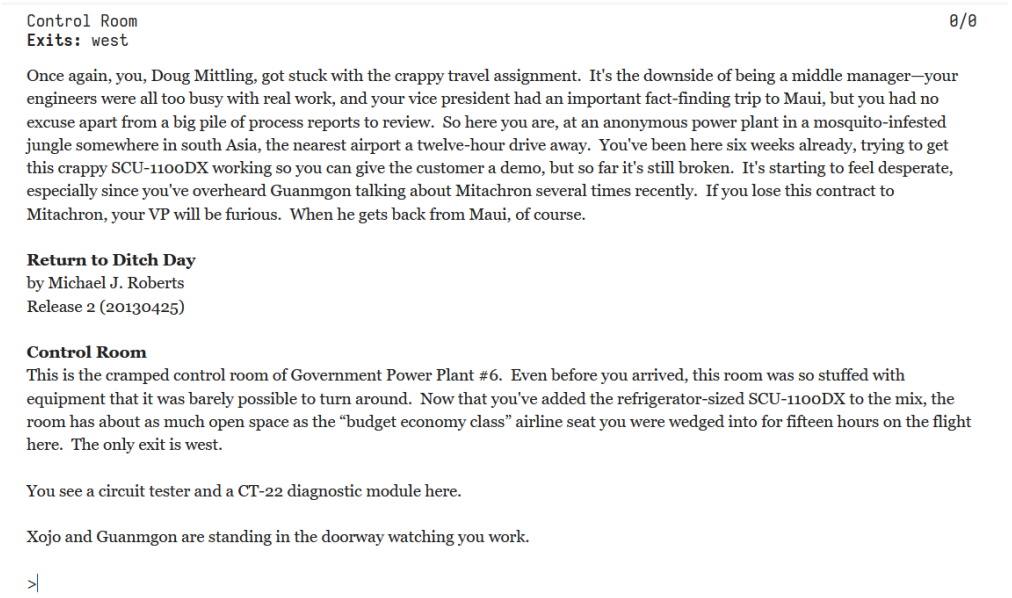

Figure 12. Return to Ditch Day, a game written by M.J. Roberts that also serves as an example of how to use TADS to create games. [58]

ZIL, short for Zork Implementation Language, is the programming language developed by Infocom, the game company who made interactive fiction popular. While the company does not exist anymore, the language is still in use by interactive fiction community, and a modern compiler, called ZILF, written by Tara McGrew and released under GNU General Public License 3, exists for compiling ZIL code into games. The compiled games require interpreter, but they use the same format Inform games do, so the same interpreters can be used for both types of games. [59]

3.2.2 Choice engines

ChoiceScript is a simple language created by Dan Fabulich for writing choice-based interactive fiction, used for games published by Choice of Games, Hosted Games, and Heart's Choice. It is designed to be easy to use even for authors with no programming experience, although there is still a learning curve for using it efficiently, and it can be used to create very complex games as well as simple ones. It is not provided as an open-source system, but it is

free to use noncommercially. Commercial releases need to be published via these companies, or by purchasing a commercial licence which requires giving the company a cut of profits. ChoiceScript games are written as text files, which are then compiled to a game executable that uses JavaScript to provide the game functionality (Figure 13). [60, 61, 62]

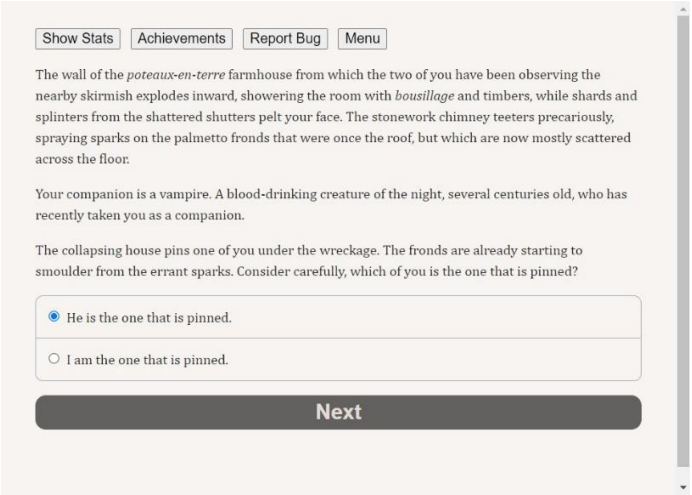

Figure 13. The game window of Choice of the Vampire, showing how ChoiceScript presents selectable options for choices to the player. [63]

Dendry (Figure 14) is a narrative engine for hypertext interactive fiction, originally developed by Ian Millington and later revived by Autumn Chen. The system is released under the MIT license. It is relatively simple, although it lacks the documentation the better known and more widely used engines have. Building games with Dendry can be done using a web service built for the purpose, but it can also be installed locally with Node.js. Finished Dendry games can be built to HTML/JavaScript pages using Node's in-built tools. [64, 65]

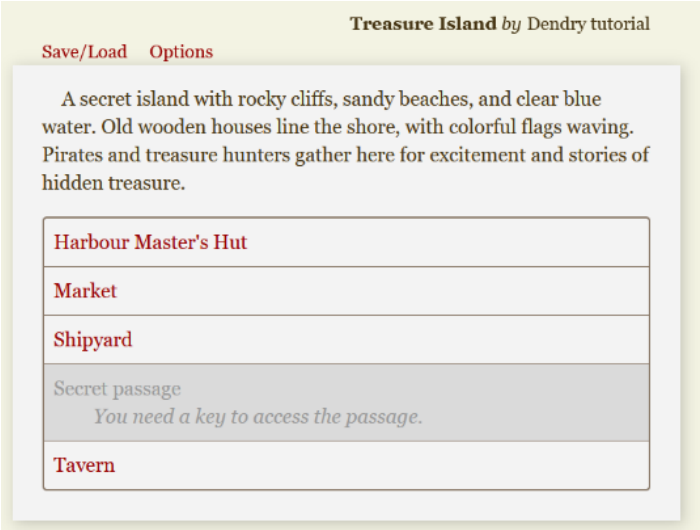

Figure 14. Treasure Island, the game used in Dendry tutorial, showcasing the interface and the look of the option list in a built Dendry game. [66]

Ink is a narrative scripting language for choice games, developed by inkle and released under MIT license. It is easy to get started with, but powerful enough to provide complex structures and programming for narratives. While ink can be used as of itself to make text-based games, it is built as a middleware to provide a narrative engine inside a larger game engine (Figure 15). Plugins for integration with Unity and Unreal are provided, but using ink as a part of other programs is not limited to those. [67] There are also third-party plugins available for integrating ink with other programming languages and game engines [68, 69, 70, 71, 72].

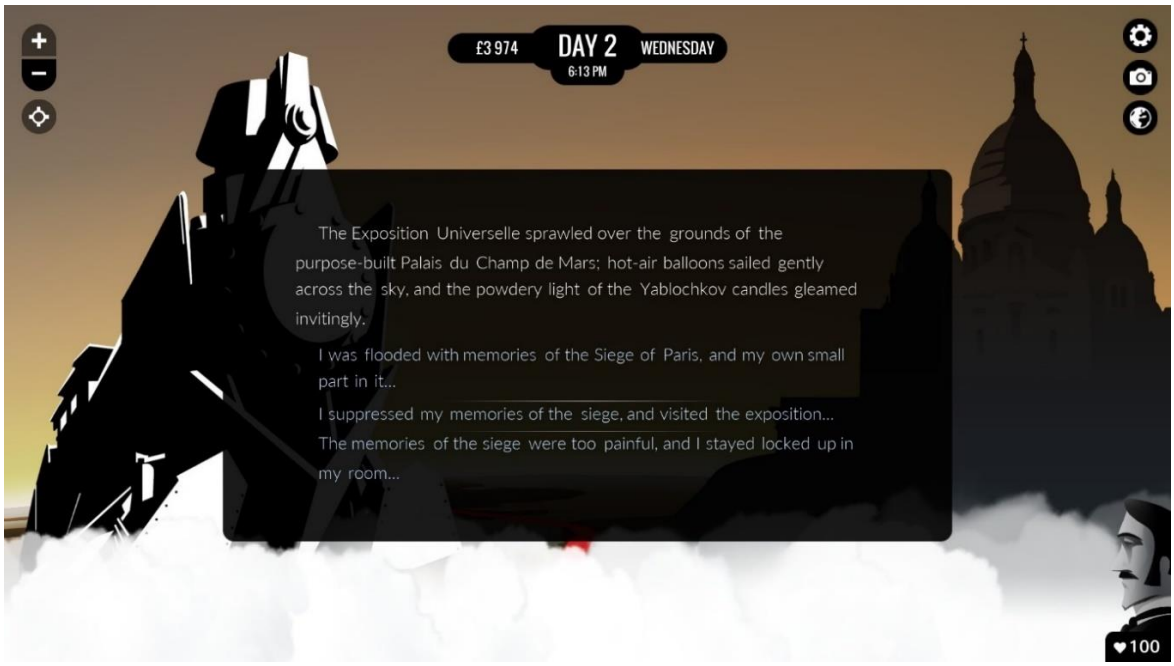

Figure 15. Under the graphical interface, 80 Days is using Ink as the narrative engine. In some places it is clearer than others. The text screens with choices are obvious interactive fiction, the map view with running clock and the inventory management less so. [73]

Twine is an open-source authoring system for choice-based interactive fiction, created by Chris Klimas and released under GNU General Public License version 3. It provides an editor in which the games are created (Figure 16), the file format for working files depending on which version of the editor is used. Twine 2 uses HTML, and the old version of the system, Twine 1, used Twine's own programming language Twee. Additional tools for compiling Twine games written in a text editor instead of Twine's own editor are also available. Different "story formats", each with their own license, define what kind of functionality and programming is available for the project. All of the stories are outputted as HTML, so they are playable in any sufficiently new web browser that supports the markup used, regardless of the story format that was used. [74, 75, 76]

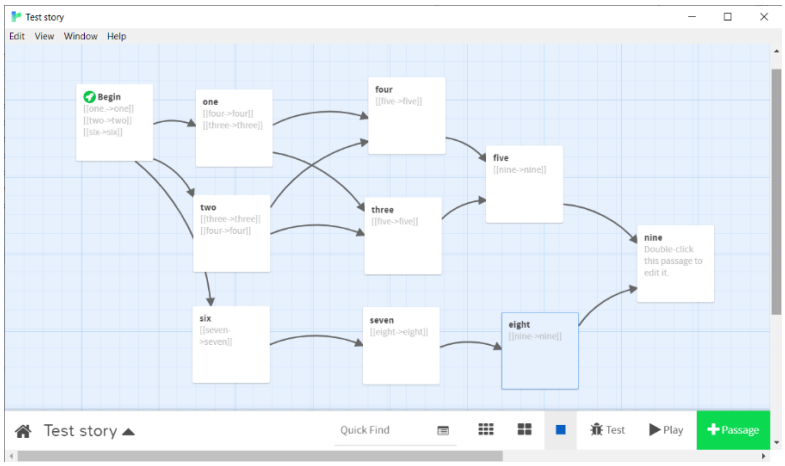

Figure 16. Twine's editor uses a flowchart-like graph for managing the game structure. All nodes present pages, and connections are links between them. [77]

3.2.3 Other engines

Binksi is a small game engine for making small graphical games for web, using tiny pixel graphics combined with ink as narrative engine. It is open source and free to use for individuals and non-profit organizations, license terms forbidding its use for companies. The editor, as well as the exported games, are built to work in a web browser. Binksi is based on another engine called bipsi, which in turn is inspired by another engine called bitsy. [78, 79]

Bitsy looks almost indistinguishable from binksi, but it uses different codebase, and the two engines are not compatible [80]. Bitsy games are composed of small rooms, where the player character can walk around and interact with different parts. The engine uses very limited graphics consisting of 8x8 pixel grid and a three-colour palette, although different rooms can have different colour palettes. Bitsy has an editor that works in a web browser (Figure 17), and completed games can be exported to HTML files. [81]

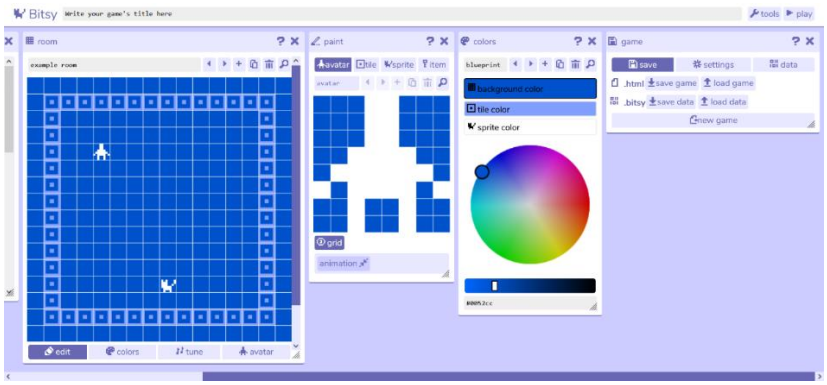

Figure 17. The bitsy editor. [82]

Quest is a parser-choice hybrid game authoring system by Alex Warren and released under MIT license. Its editor (Figure 18) includes both a graphical interface and code editor, which allows making a game for both programmers and non-programmers. A lot of functionality is in-built, but programmers can also override and extend existing functions. Completed Quest games are built in a special format that can only be played in the editor itself and a specialized web-runner, but the code of the game itself is simply xml. [83, 84]

Figure 18. The Quest editor has a graphical view of building the map of the rooms for the world the game takes place in. There are also sections for objects and characters. [85]

Ren'Py is a popular visual novel engine that can run on and build for multiple platforms. It is open-source and free for commercial use, being released under MIT license although parts of it use GNU Lesser General Public License. The engine is written in the Python programming language, and while the games are generally written in Ren'Py language, more direct Python program code can be used to extend the game's functionality. The editor for developers (Figure 19) can export games as web applications and stand-alone executables for multiple operating systems. [86] Unlike with most interactive fiction engines, images usually take a major part of Ren'Py games, but it is also possible to use it for a purely text game.

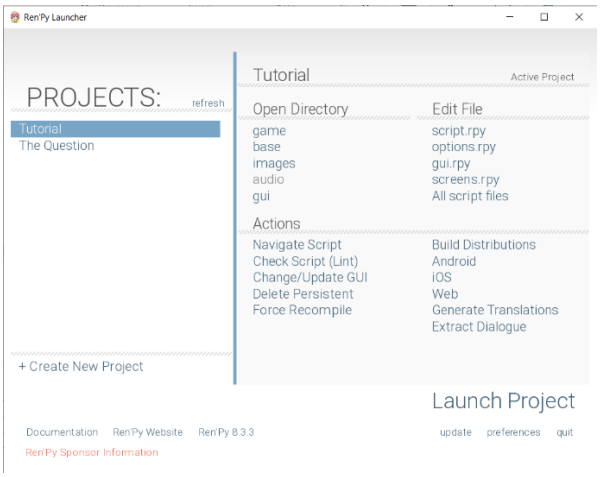

Figure 19. Ren'Py editor serves as a dashboard for a developer. It can be used to create a project, lists current projects created that way, and provides options for playtesting and exporting the game. While there are also options for opening files for editing, the editing itself happens in an external editor. What editor the file opens in is configurable in the launcher. [87]

3.3 Text-to-speech and speech-to-text technologies

Text-to-speech is a form of speech synthesis that takes written text and attempts to convert it to audio that is understandable to humans as speech. Speech-to-text is the opposite, where computer takes audio as input and attempts to translate it into text. They can be used separately or together, and most operating systems have some solutions available. Text-to-speech specifically is very useful for screen readers, which are software that reads text and other data specific for functions of screen readers, taking input from what is viewed in the active program, and converts it to speech. Speech-to-text can be used with voice recorders to automatically transcribe the recording, on automated call centres, for hands-free control systems, and to help learning pronunciation in studying languages, among other things.

3.3.1 Speech synthesizing systems

Azure AI Speech is Microsoft's cloud-based, artificial intelligence -powered speech service. It can generate speech from text, translate speech to text, translate speech to speech, and more. Unlike its predecessor, Microsoft Speech, Azure AI Speech requires an internet connection, apart from very limited conditions where it can be embedded. It also requires an active subscription for the developer, although it can be used for free to some degree. [88, 89, 90]

Emacspeak is a speech interface for Unix computers, designed to help visually impaired people to use their computers independently and efficiently. It is an improved version of a screen reader, called a virtual desktop. In translating the contents of the screen, it reads aloud the information that is relevant in the underlying structure instead of everything on screen, for example telling the actual dates instead of a series of numbers in a calendar. [91]

ESpeak NG is an open-source speech synthesizer. It supports more than 100 languages and accents, and can be run on Linux, Windows, Android, and other operating systems. It uses a synthesis method that allows languages be provided in a small size, but the resulting speech is not as natural as larger synthesizers can provide. ESpeak NG is available as a command-line program for Linux and Windows, as well as a shared library for use with other programs. It can also be used with screen readers on Windows. [92]

JAWS, Job Access With Speech, is a commercial and a very popular screen reader. It provides speech and braille output. It can recognize images, supports multiple software including Microsoft Office, and includes a voice assistant. [93]

Microsoft Narrator is Microsoft's screen reader. It comes with Windows operating systems, and thus is available to everyone using them. It provides both speech and braille outputs. The available voices depend on the version of the operating system, but they can be customized for the user's needs. It also allows navigating with only a few keys from keyboard. [94]

NaturalReader is a commercial text-to-speech software that uses artificial intelligence to generate natural-sounding speech. It can be used as a screen reader, audio book generator, and the like. With a license for commercial use, it can also be used to create audio files for distribution. NaturalReader supports multiple languages and has many voices to choose from, and it also gives options for creating new voices. [95]

NVDA, Non-Visual Desktop Access, is a free, open-source, and portable screen reader for Windows 8.1 and newer. It supports multiple popular applications, over 55 languages and many third-party voices, and it can be used with braille displays. It can speak keyboard input, including command keys if the user so wishes, report where the focus is currently on, and what the cursor is currently on top of, which allows navigating the screen by moving the mouse around instead of using keyboard navigation. NVDA also has active community, which produces add-ons to extend the program's functionality. [96]

TalkBack is Google's screen reader for Android devices, and although it also can take spoken commands, it is mostly used with touchscreen gestures. It can tell what is on the screen and give suggestions on what kind of actions are available. TalkBack can also provide a braille keyboard, which works by tapping the screen with multiple fingers indicating the location of the dots in the braille letters. [97]

VoiceOver is Apple's screen reader that is a part of all Apple devices, mobile and computer alike. It can describe what is happening on the screen, including text, objects, graphics, and people, in spoken voice or on an external braille display. Braille input via touchscreen is also supported. Apple also provides products that provide audio descriptions for movie scenes, speech-controlled reading, and typing feedback. [98]

3.3.2 Speech recognition systems

Alexa is Amazon's voice service. It can be used to build voice-controlled smart home devices, like speakers and smart screens. It can also be used to build software that can be then downloaded and used on a device using Alexa, including a mobile phone that has relevant

apps installed. Alexa can launch other apps, open websites, play music, control smart home systems, add events to calendar, and so on. [99, 100, 101]

CMUSphinx is a lightweight open-source speech recognition toolkit. It is available to use for programs written in C, Python, and Java, although the library itself is written in C. It is also included in some GNU/Linux distributions. It is sufficient for small tasks, but for larger vocabulary tasks other toolkits are recommended instead. [102]

Dragon Speech Recognition is a set of commercial speech recognition systems from Nuance Communications. It is a dictation software that provides different products for different professions, including legal and education, for desktop as well as mobile platforms. [103]

Google Assistant is Google's digital assistant for Android devices. It can take speech commands to perform a multitude of tasks, including but not limited to playing music, sending messages, opening apps, answering questions, making hands-free phone calls, making purchases, and controlling smart home systems. Android developers can also integrate it into their apps to provide voice access. [104, 105]

HTK, The Hidden Markov Model Toolkit, is a toolkit for building hidden Markov models, primarily for speech recognition research but it has also been used with other contexts. Downloading requires a registration and accepting the license, and while the models produced with it can be used elsewhere, the code itself may necessarily not. The toolkit is portable and includes many tools for analysis and training. It is primarily made for Linux, Unix, and Mac, but it can also be compiled and used for Windows environments. [106]

Janus Recognition Toolkit (JRTk) is a speech recognition toolkit and a part of the JANUS speech-to-speech translation system. It has licenses available for both commercial use and research. The JRTk provides an environment for speech recognition researchers to developing and evaluating new methods and building state-of-the-art systems. [107]

Julius is a high-performance, low memory requirement, large vocabulary speech recognition software for researchers and developers. It can understand Japanese and English, although the software itself is language independent and thus can be made to recognize other languages if sufficient models are provided. The software is also capable of running multiple different types of recognition simultaneously. Its main platform is Linux and Unix, but it also works on a Windows environment. It has been developed as a research software for Japanese. [108]

Kaldi is an open-source speech recognition toolkit aimed for speech recognition researchers, but the license allows also commercial use. It is written in C++ and is intended to be easy to modify. While Kaldi also has a version that can run Windows, it can be difficult to get it run properly, so it is recommended to use it primarily on Unix or Linux systems. It also is not recommended for people who do not know the technology and just want a tool that works out of the box. Since the intended audience are researchers, Kaldi's documentation focuses highly on the actual theory behind speech recognition, and it requires a certain level of expertise to be understandable. [109]

Microsoft Cortana was Microsoft's voice assistant, which was discontinued in 2023. It was replaced by Voice Access in Windows 11. Cortana was, and consequently Voice access is, a disability aid that allows controlling the computer it is running on with voice commands. It can open programs and switch between them, interact with items on screen, use text input and cursor, write and edit text, and so on. [110, 111]

Siri is Apple's digital assistant, integrated into Apple devices. By default, it is accessed by spoken commands, for which it uses Apple Neural Engine to decipher. However, it is possible to disable the audio input and communicate via text, instead. Siri can be used to hands-free control phone calls and navigation programs, give commands to smart home systems, play music, send messages, set up notifications, seek information, and the like. Apple developers can also integrate Siri to their own apps. The speech recognition works with multiple languages, although it needs to be given permissions by the user. [112, 113]

Speech is a framework for Apple developers to be used for speech recognition as a part of a software. It can recognize recorded and live audio and be used to translate speech to text. Speech recognition can be performed in many languages, but some are more accurate than others. The process can be performed on device with some languages, but for the rest it is done on Apple's servers. It is advised to assume an internet connection will be required. [114]

Whisper is OpenAI's open-source speech recognition model trained on a large multilingual dataset. It can be used for transcription in many languages, although the accuracy is better in some languages than others. The dataset has been made publicly available, to let developers build applications with it and to help research on speech processing. It is a general-purpose model that is available in different sizes. [115]

4 Practical work – the system

For this thesis, a system capable of serving as an interactive fiction engine was required. For financial and security reasons, the components needed to be available to utilize for free, which ruled out commercial systems. It also needed to be possible to use without a service provider gathering data from users, which ruled out most of the speech recognition providers. The aim was thus to use as much of open-source solutions as possible, and the resulting system to be as self-contained as possible. As such, none of the readily available systems found were suitable, so a simple program to test the concept was created.

The resulting program, then, could be used completely without an internet connection, and without any requirements of installing other software dependencies, although this came with an unfortunate side effect that the system requires a computer with a microphone and a capability of running Windows programs to use. It could be built for other operating systems as well, as it is written in Python programming language, which is not Windows-dependant as of itself, but the tools that were available to use in the project for the moment did not allow creating executables on any other operating system. As such, it is not possible to run the experiment on mobile devices. This was a definite downside, since mobile devices would have been the hardware people most likely have functional microphones on in 2020s. Additionally, while the system is mostly stand-alone apart from operating system requirements, it is something to be downloaded and executed from internet from an untrusted source, which may also cause concerns in people.

4.1 System components

The experimental system consists of three components: a speech recognition library to receive commands from the user to control the program and the story, a text-to-speech program to present content and interface, and an engine to handle the narrative of an interactive story and tie it all together as a single program. It is using an external executable for audio output, for practical (and technological) reasons, since the text-to-speech software is written in a different programming language (C) than the experimental system (Python), and combining those into a single executable, while possible to do, fell out of the scope of the project. The audio output executable is also located in the tool's folder structure, and as such is not dependant on any outside software installations the user may or may not have on their computer. Therefore, the system is nonetheless portable.

The system is built in a way that can load and play multiple games, as long as they are written in the format the engine uses, saved as JSON files, and placed in the correct folder. However, for the purpose of this research and due to the system acting merely as a proof of concept instead of a product ready for commercial game publishing, only one game is available in the testing suite sent for participants. Still, the option being there affects the system's functionality, as can be seen in the section about the engine.

4.1.1 Speech recognition

Vosk is a speech recognition toolkit by Alpha Cephei that has a commercial and open-source versions available, the latter released under Apache 2.0 licence, thus making it free to use with attribution. It has language packs available for over twenty languages (including English, Duch, and Swedish, but not Finnish – the tools to train a new model if needed are provided) and can be used with multiple programming languages (including Python, Java, C#, and JavaScript) and operating systems (Windows, Linux, OSX, Android, iOS). The toolkit is lightweight, portable, and works completely offline. It can also be run on a server, if working online is wanted. [116, 117, 118]

Vosk is meant for practical applications, in which it largely works out of the box, although the accuracy of the speech recognition may not necessarily be sufficient without refining the models further, depending on how and where it is used. Many reasons can cause problems in accuracy, some of which are audio quality, speaker's accent, and the model's vocabulary. Solving the problems may require implementing better audio recording, if it is not taken directly from microphone, fixing bugs or otherwise improving the software, or adapting the models with more data. [118]

Vosk models are modular; it is possible to mix and match parts of different models relatively easily. The downside of this approach is that adding new words can be a somewhat complex process, including manually adding phonemes to a phonetic dictionary. As the speech recognition relies on an existing dictionary, it is important that the dictionary includes all the words the solution is supposed to understand. It is not necessary to remove the words the system is not required to understand for the functionality to give best results in recognizing the user's speech. The existence of extra words in the dictionary does not affect the accuracy of the speech recognition. They do still affect the *size* of the dictionary, so in cases where the file size is a concern, removing unneeded data may be in order. [118]

4.1.2 Text to speech

ESpeak is an open-source text-to-speech system, or alternatively a speech synthesizer, released under GNU General Public License version 3 It is thus free to use and distribute as long as the license is kept unchanged so that the recipients have the same rights and the source code, if modified, is also available to everyone who wants it. It has been made for Linux and Windows, although it also has been ported to other systems, like Android and Mac OS. It can do speech synthesizing for over thirty languages, many of them experimental and as such may not provide very good results. The default language is English, for which some different accents are also provided. All languages can use different voices, of which some preset ones are included, and the user can also define their own voice rules. [119]

The program itself can be used as a command-line executable, or with other programs. It is of a small size. around two megabytes with its data, works completely offline, and has an additional tool that can be used to create phoneme data to create or improve language models. Due to the synthesisation being based on rulesets instead of speech samples, the data size stays small and creating new languages is relatively easy, but this comes with the caveat of the speech generated not sounding as natural or smooth as it would be with larger synthesizers that *do* base their audio production on human speech samples. [119]

There also exists the variant eSpeak NG, a newer version of eSpeak with updated code, new features, and over a hundred languages. It is a different branch from the eSpeak project, and as such it uses eSpeak's command-line options, in addition to which it has new ones. [92]

4.1.3 The engine

ANarrator is the proof-of-concept piece created for this experiment, tying together the other parts of the program. It takes the interpretations the speech recognition translates from microphone input and turns them into commands understood by the system. Based on the given command, it then does actions associated to the command, thus allowing the user to interact with the system or, if it fails to understand the command the user was trying to give, asks for a new one. To accommodate the differences between pronunciation and what the speech recognition understands them as, some alternative ways to pronounce the available commands have been programmed in. The language models may or may not interpret speech correctly, depending on user's accent, the audio environment (background noises), and the like. To help the user have their commands work as intended, the system also prints out its

interpretation of what the user said, to help the user understand how to adjust their speech in case they cannot make their commands work.

ANarrator also acts as the game engine, directing the user through the story played. For this research, only a basic functionality was required, so the engine is not complex. The system thus is very simple: a block of text, which can also be called as a page, read as audio by the engine to the user, is followed by two or three choices, which are also read as audio, since the system does not have an actual visual user interface, apart from some output on command-line. User then selects, by voice command, a choice from the provided options, with which the engine then directs to a following part of the story, which is another page followed ty two or three choices. In the version used in this research, it is a very basic story, in which all the choices are equally valid, although some may skip some pages. The system in this state cannot deal with variables in text blocks, so all variety is done by differing branches, most of which have almost identical text.

Since the functionality of the program is limited, as it is not a general-purpose speech command system but rather a hyperlink structure in audio form, only a limited number of commands is available for the user any given time. These differ based on which part of the program the user is currently in. There *is* some text output on command-line (mostly telling the user when the program is waiting for a command and when not), but otherwise the whole interface is audio-based, so there is no way to use menu overlays the same way it would be in a graphical interface. At all times, the user can ask for interface help, ask the program to repeat what it previously said, and select from provided options (which themselves depend on where in the program the user currently is in; the options in game selection are not the same ones than in the middle of a game, for example), or switch between audio and text output. In addition to these, there are the mode-dependant commands: in a game, it is possible to quit the game and return to menu, but not to quit the game directly; in the menu, it is possible to select a game and start to play, or to quit the program entirely (see Figure 20).

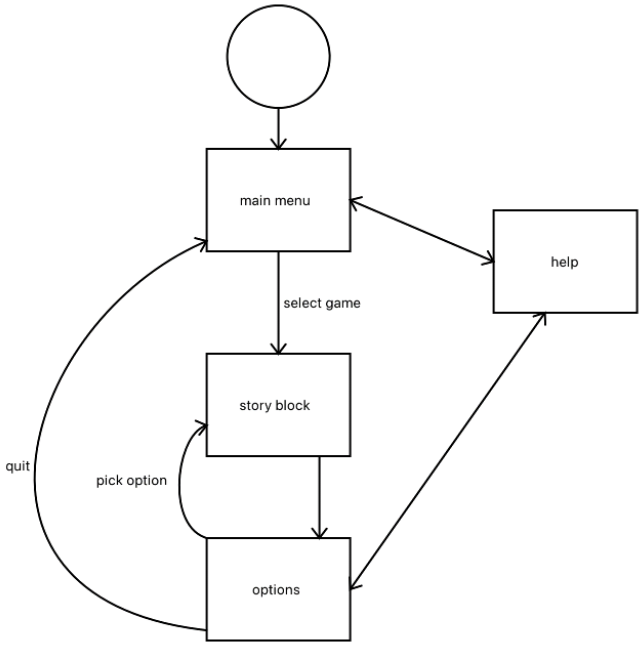

Figure 20. A visualization of how ANarrator moves from one text block to another.

4.2 Usage

ANarrator is a command-line program (Figure 21). It can be launched by double-clicking the executable, calling the said executable from command-line, or, if ran from the source, running the Python script. Apart from whether or not the window closes when the program is closed, all these methods give the same result, since the program does not take any command-line parameters. All the commands the engine understands are hard coded in the program code, so changing the available options (apart from the choices in a game, which are defined in that game's file) requires changing the program code. That is not difficult to do, however, assuming one is working with the source code and not the executable.

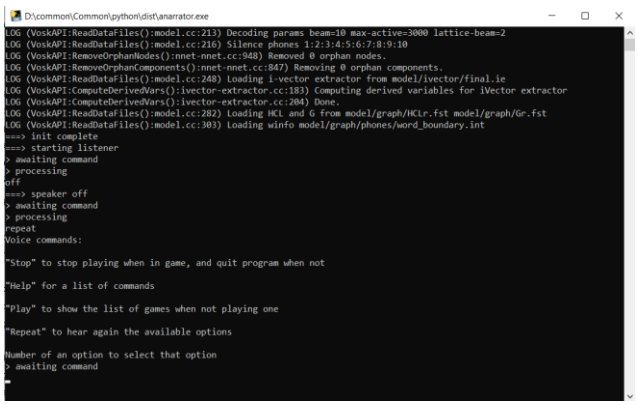

Figure 21. The ANarrator window shows important feedback to user in all cases, but the program can be switched to text output instead of audio one if needed.

The contents of the games are stored in “content” subfolder in the folder structure containing the executable, one JSON file per game. The program automatically checks that folder and attempts to parse all JSON files in it as games, so it is important to not put in it any such files that are not game files. Changing the available games requires only changing the game files in said folder, the executable does no need to be changed. The system is already built for that.

The game files must contain only one game each. Although JSON format itself would allow including multiple games in a single file, ANarrator cannot handle that. The required data structure is the following, with the lines with special purpose numbered:

```
  {
1     "title": "Title of the story",
2     "start": "block1",
3     "block1": {
4          "content": "Text to be shown on page",
5          "options": {
6               "opt": "3",
7               "1": {
8                    "content": "option text".
9                    "goto": "block2"
                }
                …
          }
      }
      …
  }
```

To clarify how all this works, the meanings of the numbered lines and their required features in the structure are the following:

1: The title with which the game is shown on the game list.

2: Points to the first block of the story.

3: The key with which the block in question is referred to: this can be whatever the author wishes. In the sample story, a combination of a "page number" and a row of path identifiers is used, so these are character combinations like "0a" and "9caab".

4: The contents of the block; this is one page of the story.

5: The options available for the block are listed under this.

6: The number of options available, can be 12 as highest. 0 means end of story.

7: The key of the option. This must be a number between (inclusive) 1 and the number of options available for the page.

8: The text of the option, as provided to the player.

9: The key to the block the option leads to.

There can be as many page blocks as needed. Number of option blocks must equal the number of options set in "opt", and their keys must equal the given scale per options block.

As the system currently is incapable of understanding variables in text blocks, all branching needs to be done by different text blocks. This is far from ideal since the amount of identical or near-identical text blocks becomes unsustainable if a game is longer. It is nearly unsustainable already in the test game, which has only ten to eleven pages, depending on player choices. In case of continuing to develop the system, this inability needs to be changed.

The text-to-speech is currently done by calling the command-line executable of eSpeak, which is included in a subfolder. Ideally, there would be no secondary executables. Although there already are Python libraries for both eSpeak and eSpeak NG, they all require an existing installation of said eSpeak or eSpeak NG on the system, which goes against the principle of ANarrator being portable. ESpeak NG could, potentially, be included in the program, either by changing the programming language from Python to C, or by building eSpeak NG as a Python library that does not rely on external software installations.

The speech recognition takes place in the ANarrator engine, using Vosk library for Python. This functions by taking audio input from user's microphone and delivering it to the library, which then gives a text interpretation of what was said. This text interpretation is then provided to the engine, which checks if it matches one of the commands coded in. If a command is found, the engine proceeds to executing it, and if it is not, new input is asked

instead. As it can sometimes take multiple tries to make the commands understood, the program prints the interpretation on command-line, so that the user can see how what they said was understood. This is to help the player to understand how to pronounce differently.

Because Vosk tries to interpret all audio it is offered, some safeguards needed to be programmed in. Breathing into the microphone gets interpreted, at least in the system where the development testing was done, as “huh” or “ha”, or sometimes just an empty string, all of which are not valid commands, and especially not when not given intentionally. Although the engine does not accept these as commands, simply being provided them to be checked is a problem, since all commands that are not recognized cause the system to ask for a new one, which does not sound like correct feedback from the system when the user did not in fact intend to give one. Because asking players to not breathe is not a sustainable course of action, the system filters empty strings and all “huh” and “ha” words without recognizing them as input. This has the drawback of not recognizing if those sounds are given as command on purpose, as the system acts as if nothing was said at all, but this is something that needs to be improved in a later version of the program if it will be developed further.

Another problem that needs to be improved in a later version of the program is how and when the speech recognition accepts input. In the current version, Vosk is always on. This makes the system being able to take audio commands at any time, but if a command is given before the options are provided, the system gets confused and auto-progresses through the choices based on whatever command was given at the wrong time. Having user make choices when they do not yet know what choices are is not the way the software should function, so the system tries to prevent that by accepting input only at times when the user should be giving it. This is indicated on the command-line window. The solution works, most of the time, but not always, for reasons testing has not managed to reveal. A better solution would be turning Vosk on only when needed, or accepting the input at wrong time but doing something to check with the user if they indeed intended to give the command or if it was an accident, or alternatively allowing only certain types of commands given that way, so that user could, for example, close the game or program in the middle of a sentence, but not pick an option as to how to progress the story.

5 Experiment

The data for this experiment was gathered by having the testers play a game they knew nothing about, and then answering questions about their experience of playing. The game tested was an interactive fiction story using an experimental audio interface, instead of the usual text and mouse/keyboard-controlled one, that was made for this specific purpose. It was also a story the testers should not have had knowledge of before, thus ensuring that whether or not they were able to follow it was not based on their prior knowledge on what was happening, and instead on how the program was capable of communicating it.

The experiment was run with two different tests. First, testers were recruited anonymously online, where they were provided introductions and a download link to the game. When this method failed to gather enough testers, and as such also enough data to draw any conclusions other than what might have caused the lack of testers, a secondary test of recording live sessions and observing tester behaviour instead of relying on statistical data was conducted. Both tests used the same version of the game and the same questionnaire.

5.1 Questionnaire

The questionnaire was built to find out not only how users feel about the tested system, but also how certain background details of users may or may not affect the experience. However, due to the lack of test subjects, most of the questions in the questionnaire ended up being useless in that regard, since there simply was not enough data to do any kind of conclusions on the subject. That means these questions remain unanswered in this research, although they are something that would be interesting to conduct more research on later. How does user enjoyment about audio-interface interactive fiction depend on user being already familiar with the concept of interactive fiction? Does the speech recognition have more problems with users who speak with some non-native English accents than others?

The questionnaire consists of three types of questions, which are briefly described in this section. The first type is the questions about the background details hypothesized as possibly having some effect on the user experience, the second type is about how the technical aspects of the game were experienced, and the third type is general feedback. The full list can be found in Appendix A, in which the full questionnaire is provided, including the exact wording of the questionnaire and the answer options in the multiple-choice questions.

In the background section, there is a question about how much the user has previous experience with interactive fiction. This is to find out how having previous knowledge of the genre affects the user's experience with the voice-controlled format. The hypothesis in this is that users with more experience would have less difficulties in understanding how the program works, and as such they would interact with it more easily. Another question in this section is about user's native language. This is to find out how this language affects the accuracy of the speech recognition in use, although it uses English. The hypothesis is that, since the speech recognition does not correctly recognize all pronunciations and since a person's first language often affects how they pronounce other languages, the users from some languages would have more problems than users from others. Since languages have different dialects, and the correlation between pronunciations in different languages is not so clear, it is also equally possible that this detail has no such effect. The findings were not sufficient to draw any conclusions.

In the technical section, there are questions about how much users had to repeat commands, whether they could follow the narrative, whether or not they had to quit the game without completing it, and so on. This is to find out the details detailed in the hypotheses of the background details' affects, as well as finding out how the program is performing overall, since it was, as stated, created for this specific purpose, and as such, is essentially a prototype.

In the general feedback section, there are open-ended questions about what works and what does not, which were meant to provide a clearer picture of how the testers experienced the system. They could also be useful if the program is developed further, combined with the technical details about what parts of the program were causing problems to the testers. Although making a good product was not the focus of this research, there is no reason why the problem areas pointed out in the feedback could not be improved based on the data.

The questionnaire was done with Google Forms, which automatically draws some diagrams from the data. To preserve the anonymity of the answers, the testers were advised to answer without being logged in to their Google account. While that can mean the same person could send the form multiple times, there was little reason to believe that people volunteering as testers for academic research would do so. Given the small number of answers, it appears they indeed did not do that.

5.2 First test

The first test was performed online, in the form of an anonymous survey. The participants were recruited via multiple forums and Discord channels that were deemed to potentially have members who might be willing to participate. The goal was to gain a wide variety of people, where some would have previous experience with interactive fiction while others would not, which would give data on how the testers' knowledge on how interactive fiction works might affect how they interact with the product and how they feel about it. Another goal was to recruit people with different first languages, which would then also give data on how the speech recognition used performs with different accents and how the user's first language affects the accuracy of the speech recognition and the frustration of players who need to try multiple times to get their commands recognized, or who get their commands recognized as *different* commands than what was intended.

The recruitment was similar in all platforms where it was attempted, the differences in how the specific platform allowed the presentation notwithstanding. A forum post or a chat message detailed what the requirements for participating in the test were, so the potential testers could know beforehand if they had the required equipment, what the test entailed in general, so that people could decide whether they were willing to participate or not, and a link to the packaged file that contained the software itself and a readme file giving further details on what to do in the test, how to use the program, and a link to the questionnaire.

The requirements for participating were a computer capable of running Windows programs (preferably Windows 10 or higher, since that was where the program was built and tested and as such, known to run on), a functional microphone on said computer (headset recommended, since the background noise, including the computer's cooling systems and hardware noises, and the user breathing, was confusing the speech recognition, and headsets usually filter such noise out more than computers' in-built microphones), and willingness to run the software and participate in the test (and trust an executable from unverified source enough to actually run it). It is unknown how many testers did download the program but did not answer the questionnaire, for whatever reason, but all verifiable data seems to point out that the testing package did not get many downloads, so antivirus programs flagging the executable was probably not a very big problem in keeping the number of answers low.

The readme contained the same general information the recruitment post did, apart from having a link to the download, which would have been redundant since it would have had to

already be downloaded for the tester to be able to access the readme. In addition to that, it explained how exactly to perform the test.

Detailed descriptions on how to use the system and play the game were not provided, so that it would be possible to get data on how self-explanatory the audio interface is, but the commands to use the program were provided as a reference. Even though the commands were also available via the program, being able to see them instead of needing to try to remember them from pure audio decreases the cognitive load of the user and as such, giving a reference list seemed like a good idea. It was also meant to decrease the user's need to use the "help" command, which was a command in the game that gave the list of commands (in spoken format) available, and all in all help the user to complete the test on their own. The program also includes a couple of unlisted commands for testing purposes, but those were not included in the readme instructions, since they were not intended to be used in testing.

The readme also included information about potential problems the user might encounter with the program, since it was a prototype and might act weirdly or at least opposed to user expectations in certain situations. Sometimes it could take a long time to process which might cause the user to enter command multiple times thus further confusing the system, causing more delays, and frustrating the user further. Sometimes, it might simply not understand a correct command because it failed to interpret the pronunciation. All of these could hinder the tester's capability of completing the game, and since the user could not ask from someone who knew the program better, this information was provided as a list of known issues to prepare the tester in case they encountered any of them.

Finally, the readme included the URL address to the Google form where the questionnaire to be filled was located. The instructions also mentioned it should be filled after finishing the playing, and that it should be filled without being logged in to a Google account in case to prevent accidentally collecting data that would make it less anonymous. Although it was not verified whether or not that actually happens, there was no reason to take the risk and contaminate the data. Also, because part of the rationale behind selecting the tools to use for building the game was to not use anything that would collect the user's data to some server with or without their permission or even knowledge, it seemed like an appropriate precaution.

This test did not gather a sufficient amount of data, since only a couple of people participated in it, and as such, another approach for testing had to be considered.

5.3 Second test

After the first test failed to provide sufficient data, a second test was conducted. The core of the test was same: a test subject playing the game for the first time. The game used in the second test, also, was the same than the one used in the first test, but unlike in the first test, the session was observed and recorded, instead of testers conducting everything by themselves. For this reason, the data could not be collected completely anonymously, unlike in the first test. Despite this, ensuring that the identity of the test subjects was not included in the data was done as fully as possible.

Recruiting the testers was done with as little details as possible, to ensure the reactions in the testing sessions were authentic. The testers were told that they would be playing a game that required using a microphone and speaking and understanding English, that the session would be recorded, and what were the technical requirements to run the game (and participate in the test) to not accidentally pick testers who could not participate in a testing session that required them running the program on their own computer.

The testing sessions started with a debriefing, in which the testers were told further details of what the test included and what was expected of them. To prevent them from investigating the program beforehand, so that the testing would authentically be about them interacting with it for the first time, this was when they were provided access to it. They were told it was an audio-based game that was controlled by voice commands, and a command list was introduced, but they were not told how to navigate the program other than giving the list of commands it accepted, nor were they told how to actually play the game. They were then advised to verbalize what they thought during the test when trying to communicate with the program, in a manner of thinking out loud as they did the testing.

The testers were, for the most part, told to try and figure out the system themselves, although if they got stuck, or forgot the commands, they were allowed to ask for help. In those cases, the advice from the observer was meant to be as minimal as possible, only as to allow the tester to be able to figure out the solution themselves, or allow them to remember the commands again, potentially just the “help” command in the program, which was included there for that specific purpose, since the interface was only audio and as such there were no visual reminders of what the commands used by the program were.

The testing itself was recorded by a software capturing what happens on the screen and through the computer audio. The thinking out loud method was necessary to find out what the testers were experiencing during the test, but since the software that was used in the test was controlled by voice commands, actually thinking out loud would potentially have confused the system. There was no way to have the speech recognition know which audio input was intended for it and which was not, so speaking unrelated words that then would have been picked up by the speech recognition would have caused plenty of errors when the program would have interpreted thinking out loud as valid commands and failing to do so, or in some cases, made the program to select an unwanted option because it interpreted something unrelated as a command it *understood* as a command. For this reason, the testers were instructed to think *in writing* instead, in a separate window from the program tested, and the contents of this window were then captured in the screen recording alongside what was happening in the window of the program tested and what the audio input/output was. The latter one could only be seen by the feedback on the test program's screen, but it was included in the recording of the audio. That was necessary to see how well what the speech recognition interpreted matched what a human observer heard being said.

Debriefing had the testers answer the same questions the questionnaire in the first test did. To preserve the anonymity in the questionnaire answers, the questionnaire form itself was not used for that purpose, but the questions were same. The testers were also asked if they had any other comments on the experience that were not covered by the questionnaire, although the question list was designed to cover as many aspects of the test as possible, including open-ended questions that were meant for other comments the tester might have.

The data from the post-test questions of the second test was then added to the data from the questionnaire from the first test, to track what the general feelings of the testers about the concept were. While the results were still too scarce to make any generalizations about the interest in such a product, some conclusions about the technical aspects could be made, even if it was not to the degree the original purpose meant for the research intended. The comments in the questionnaire were added to the observations from the recorded sessions to see how users interacted with the system and to see what the testers were struggling with, to help identify what the problem areas in the program were, to see where to direct potential further research on the subject, and how to improve the product meanwhile in order to make interacting with it more engaging and fun.

6 Results

The goal of this thesis was to find out whether or not a fully voice-based interactive fiction format would be considered interesting. In addition, there were secondary goals about how familiarity with the genre affected that opinion, as well as finding out how different background details affected the user experience with the proof-of-concept work.

The questions this thesis was interested in, thus, were the following:

- Was playing the game enjoyable?
    - Did the player's history with interactive fiction affect whether or not the game was found enjoyable?
- Were the players able to follow the story?
    - Were the difficulties in following the story caused by the audio format itself, or was the low-quality text-to-speech system a culprit?
- Was the game able to communicate how it works?
    - Was the format easier to understand for players with history with interactive fiction than for those without?
- How difficult it was to control the game with a speech interface?
    - How did the user's first language affect the difficulty?
- How did the system work?
    - Would it be worth improving further, and how?

The turnout for test subjects was poor, so the amount of data gathered was not enough to generalize results and gain proper insights on the subject. Although the research questions remain unanswered, the tests themselves provide some data that can be used as a basis for future research. In addition to that, the responses seemed to confirm some of the initial assumptions about the system's functionality, so they are also worth considering in deciding how to improve further testing.

6.1 Results about recruiting testers

From the poor turnout of testers, it is obvious that the recruitment process did not work. There are multiple reasons that could have caused this, all of which need to be taken into account for planning further research in order to ensure a more successful result.

6.1.1 Platforms for recruiting testers

The recruitment for testers took place on multiple platforms on internet. Some of these platforms were discussion forums, some were Discord servers. This was presumed to provide a large enough audience to attract testers, but something clearly was wrong with the strategy. It is possible that the selected platforms simply did not have enough people who were interested in testing, but some of the locations are specifically directed to people developing interactive fiction and testing the work-in-progress games, so while that assumption may be true for some of the platforms, it certainly cannot be the only explanation.

It is possible that in future research, focus should be more on interactive fiction communities, to provide more testers. That, however, does not help in gathering the comparison data from people who do generally not play such games, which was one of the original research questions and the reason the platform focus was not *solely* on interactive fiction forums.

6.1.2 Technological requirements

Participating in the test required downloading and running software from internet. While some people are willing to do that, others are not, especially from an untrusted source. There is no proof this caused people to not participate, but it is nonetheless a possible explanation. Selecting platforms where users are more used to downloading program files could be a possible solution to get more testers, although that could also make the project be less visible. Some more marketing would be required to not have the project lost among others too much.

The software itself was a Windows executable, so it, too, presented certain requirements. The test could only be performed on Windows computers, or by players who know how to run Windows programs on other operating systems. It was not possible to run the test with a phone, which is what many of the users on the platforms where the recruitment took place would have used. This may have ruled out testers who otherwise would have participated. The solution to this would be to make the test web-based, but to maintain the original design

principle, it would also be important to make sure the speech recognition itself would not require a connection to a server.

Yet another technological requirement was the need for microphone for running the test, since that was the input method for the system. This requirement did provably have an effect in the tester turnout, since some potential testers informed that they would have participated otherwise, but they did not have a suitable microphone on their computers. Presumably they would have had one on their phones, so the solution for the previous point, making the test web-based, would also apply to this one, with the addition that it would need to be possible to run the test with a *mobile device*, not only with a computer. This additional requirement would also need to be considered when building a web-runnable executable.

6.1.3 Reaching potential testers

It is possible that the recruitment posts simply failed to be viewed by enough people. Solving this would probably require more advertising, but doing that without resorting to behaviour that would be considered spamming would need some further planning. One solution would, of course, be posting on more platforms, assuming suitable ones can be found. Presumably, focusing on interactive fiction communities would provide a more reliable source of testers; this would not provide more comparison data for the possible differences between players who play interactive fiction and players who do not, so it would not work as a sole solution. While interactive fiction players most likely would be the ones interested in a different form of interactive fiction, finding how understandable the system is to people who are not used to the genre is an important research question.

Perhaps some kind of reward should be provided for testers, although how to do that if the data is gathered anonymously is another question, as well as how to finance it, since a simple “thank you” message presumably would not be enough. Another possible, and perhaps more viable, solution would be setting up a testing booth in some kind of event. In that case, passers-by could participate in an observed anonymous testing without any hardware requirements from themselves, and that could be sufficient to gather more testers. It might have more requirements for the hardware brought in there to use in testing than letting people run the program on their own devices would, which would need to be taken into account.

6.2 Results about the test itself

The test itself, when performed, seemed to mainly function fairly well, or at least the data gained from it appeared decent. Some issues worth of note appeared when running the tests, that may also be worth consideration if further research is conducted.

6.2.1 The questionnaire

The questionnaire appeared to mostly be understood by the testers. One question, however, seemed to be confusing, or perhaps the experimental system's appearance was confusing and that reflected in the answers. In any event, the question about potential improvements for the system was answered by some testers with the full output on the game window as if it was an error log, instead of the intended user feedback about what they would like to see changed. It is possible that this was intended as a shorthand to tell they wanted to see the output gone, but since they did not state their intentions, it can only be interpreted.

Due to the unreliability of the speech recognition, getting rid of the output altogether would probably decrease the user enjoyment. Without output showing how the input was interpreted, the players would not know what was wrong with their commands. Combined with the fact that the processing could occasionally take a long time without any indication on whether or not anything is happening, the users would probably try to give the same command again and again, thus further confusing the system. The solution for this, of course, should be to improve the accuracy of the speech recognition, not annoying the players, but until that is achieved, letting the players know what is happening seems like a more desirable solution, especially for testing purposes.

For further testing, the question about improvements would probably need to either be reworded, or otherwise be given a further explanation. Apart from that, all the answers in the questionnaire seem to measure what they were supposed to, although it naturally is not possible to know what the testers were thinking while filling them. The questionnaire could otherwise presumably stay the way it is. If more questions would be thought of, they should of course then be added.

6.2.2 Conducting the test

The testing itself, while doing its job, proved rather clunky in practice. Further tests might benefit from being improved in this regard. As mentioned in the technological requirements, a

version of the software that can be run on web might be beneficial, to remove the hardware requirements apart from microphone. Making a version that can be run on mobile browsers would allow testers to use their phones, thus making testing easier for people who primarily play interactive fiction on mobile devices. It would also presumably allow cross-platform compatibility, since there would be no operating system -specific executables.

Using speech interface causes its own problems in observed sessions, because the speech recognition does not know when something said is directed to it and when it is not. This means that any discussion or off-handed comments done during the test are going to be picked by the speech recognition. In worst cases, this can result in the program performing an unwanted action. Since the speech is necessary for recording the test, muting the microphone for the duration of the test is not very easy. It works only if the audio for the session is recorded separately in a test where both the observer and the observee are in the same room, or in a web session, when the audio input comes from a separate device from the one where the game is run. A testing setup with a three-person video conference, where the tester is one person on their device, and the observer is on the conference via two separate devices, running the software on a computer and relaying audio commands via phone, might theoretically work to some degree, but this was not tested.

One solution to the talking problem is to try to avoid speaking during the test, and instead communicate via text. This probably works better with a video conference test, when both the observer and the tester can communicate via chat, than in a test where both are in the same room. Another solution would be to add a function to the game that allows it to pause listening when needed, so that normal communication during test would be possible. This would require changing the design principle of the program so that it would also accept keyboard input. Given the circumstances, it sounds acceptable.

6.3 Results about the format and the system

While the data was limited and as such the research remains inconclusive, it seems like further testing could be committed. The low quality of the proof-of-concept system affected the enjoyment the testers could gain from it, and they would not be interested in playing it again without improvements, but none of them rejected the format outright. However, all of the testers also ended up being people who play plenty of interactive fiction. It is more likely that the people who would end up playing audio interface interactive fiction are people who

otherwise also play other kinds of interactive fiction, but the data from how they appear to people who do not normally play such games would still be interesting to see.

From the data available, it seems the major problems largely were the ones that pre-testing the software indicated. Technological issues, due to the lightweight speech libraries, were the cause of most reported problems. With some issues it is not clear whether the cause was the low-quality audio causing difficulties, or if the audio interface format itself was at fault. More data could possibly have cleared that out, but it would be more reasonable to improve the current system before further testing than to run tests on a larger group just to find out how the problems correlate with each other.

6.3.1 Audio output

Every tester reported the synthesized speech as something that needs to be improved for the format to be enjoyable. In future work, it would be best to try to improve the speech synthesizing quality before conducting more tests, as it could be the cause of certain other problems that were detected during the testing. And even if those other problems are not connected, improving the speech quality is still preferable, since it would enhance the user experience by being more comfortable to listen.

Given that the system had to be made to include a command to turn the audio output off simply because listening to it during development proved to be too annoying, the user experience about how grating it is does not come across as a surprise.

Some feedback suggested using pre-recorded human speech instead of text-to-speech. That would certainly improve the audio quality, but with the downsides of increasing the file size and sacrificing the flexibility. For the original design principle, which was to have a lightweight stand-alone engine anyone could write a story in, the suggestion would not be suitable. However, it would work well in audio-interface interactive fiction in general.

Some of the feedback also suggested adding environmental and atmospheric noises in the audio output, thus essentially turning the game into an interactive audio drama. This sounds like an interesting idea that would go well with the prerecorded human voices. It is something that would need to be used carefully, in order to not make hearing and understanding the actual speech output harder, and to not confuse the speech recognition.

6.3.2 Speech recognition

Speech recognition was constantly reported as a problem. None of the testers managed to play the game without having to retry stating commands multiple times. How much this was due to background noise disturbing the speech recognition, and how much the cause was testers' pronunciation, is not clear. Both caused problems with the program during the development, so the result is not a surprise, even though the root cause is not clear.

Speech recognition accuracy is clearly an issue that needs to be addressed, if the product is going to be developed further. There are multiple ways it could be done. Switching to a more accurate model could be possible, although it would not be as lightweight. Switching to another speech recognition software that would have larger support for different accents is another option, although the open-source options are limited. It might also be possible to update the language model used by Vosk to recognize said different accents. The engine uses very limited vocabulary, so it might be possible to have a lightweight model that supports many accents, but only understands a few words.

6.3.3 General usability

Following the story caused problems to some testers. The cause of this is not clear. It is possible the speech generation was causing the speech to not be comprehensible for some players, or it was too grating to listen and they spaced out. Or perhaps the speech was too slow, or the need to constantly repeat commands too distracting, and the players forgot what was happening. It is also possible that the players simply were not able to follow the spoken English well enough, and the quality of the speech interface was not the main problem.

Users also had to multiple times use the "repeat" command, which repeats the latest text block. This may be connected to the previous issue, since not being able to follow the story could certainly cause the need to listen to the page again, and listening to the same page multiple times might lead to the player forgetting what was happening on the page before. It is unclear if this is connected to the player not being able to understand what the text-to-speech utility said, if the text was too long to remember, or if they simply lost focus while listening, possibly because the speech was too slow or too grating to listen.

Although the reason is not clear, both of these issues could be a result of the poor quality of the speech synthesis. Improving it might help getting rid of these usability problems, or at least finding out if the cause is somewhere else.

6.3.4 Other results

The responses were generally positive, despite the problems. The system was described fun, but requiring improvement. It seems that at least the interactive fiction audience, which is what most of the testers consisted of, would be open to audio interfaces. Of course, the test group was limited, so a larger sample is needed to draw proper conclusions.

One of the testers expressed surprise about the game actually talking, despite being told beforehand that the program had audio output. This was an unexpected outcome. It is unsure how to prepare users to having a talking program, if after directly telling them they are still surprised. Luckily, this caused no problems regarding the tester's ability to test the game.

The proof-of-concept version of the software is following a choice-based structure. Some of the feedback suggested that a parser-based model might be a better fit. Since giving commands by speaking certainly resembles giving them by typing, as parser games usually do, this does sound like a natural direction. However, some people enjoy choice-based games but not parser-based games. Preferably, both modes would be supported.

6.4 In summary

The research remains inconclusive, but it is still not necessarily a failure. Interest in, or at least curiosity about, audio-interfaced interactive fiction exists, and further research could provide more answers. Some questions set in this thesis, like the ones about speech recognition accuracy, fit better in research about those subjects, although the answers would have been interesting to see in the context of the proof-of-concept system.

It is clear the tester recruitment was a failure. For further research, it needs to be done better. Improving the system to have less hardware requirements may help. It is also clear the system needs to be otherwise improved as well, before it would be used in further research. Its current proof-of-concept state is not satisfiable, nor is it suitable for easy testing.

7 Conclusion

This thesis set out to find how a fully voice-interface interactive fiction format would be received. Using an experimental proof-of-concept system for an experiment, a test to find the answer was conducted. Unfortunately, the experiment failed to attract enough testers, and as such it failed to provide sufficient data to draw conclusions on anything else than the performance of the experimental system. Therefore, to answer the research questions, more research, with a better functioning product, would be required.

7.1 The current system

The current version of the proof-of-concept serves its purpose, but not much else. It is lightweight and, apart from some behavioural oddities, should be easy to use. It allows adding and removing stories easily, but the stories themselves are not very flexible. The engine can handle branching, but it does not support variables inside text blocks to provide minor variations in text without full separate branches.

The biggest weaknesses in the system are the quality of the speech output, and the inaccuracies in the speech recognition. The text-to-speech software, eSpeak, seems to produce better sound for single words and short phrases than longer texts, so it is possible that cutting text to play words separately instead of full paragraphs would solve the problem. The largest problem with the speech recognition is that it is constantly on, which causes it to catch sounds not intended as commands for it, and this can cause the system to get confused. It can also be inaccurate in trying to decipher what the user tried to say.

While the executable used in testing requires a Windows computer to run, that is not a requirement by the source code but caused by the environment in where it was built. Someone with access to a different operating system would also be able to build an executable that can be run on that particular operating system.

7.2 Future work

The proof-of-concept systems used in the experiment was serviceable for the purposes of this thesis, but it could be improved in multiple ways, some of which are based on features that were cut in order to provide a functional prototype for testing, and others that were inspired by the results of the testing. Of course, completely new systems could also be built.

The primary goal for improvement should be to improve the speech synthesis. The proof-of-concept system uses eSpeak's command-line executable, that functions without installation. While this was sufficient for the prototype, it is also very limited. For better functionality, it would be better to integrate the speech synthesizing into the same program. That would also allow upgrading it from eSpeak to eSpeak NG, which is the newer and improved version. This would require changing the system's primary programming language from Python to C/C++, but that can hardly be seen as a downside.

The other part of speech interface, the speech recognition, does also require fine-tuning. It is presumably fine to keep using Vosk, although the code for taking the actual speech commands requires rewriting. It would probably be best to start attempting to increase the accuracy by working on the language model, instead of making any drastic changes. Other options could also be considered, if other good open-source solutions are found.

For testing purposes, it would probably be best to distribute the testing software as a web executable, on a platform like itch.io where it could be easily played with multiple devices and link to it could be provided to the testers.

Finally, the narrative engine. The proof-of-concept system uses a very limited structure, which is not suitable for larger games with a lot of variation. To be a viable tool for interactive fiction, it would need to incorporate a better narrative engine. Ink would be a good option for a choice-based system, but there are also other open-source choice-based options that could be used. If a parser-based system would be added, Quest with its support for both choice-based and parser-based games could be worth considering.

References

[1] R. Ziegfeld, “Interactive fiction: A new literary genre?,” *New Literary History,* vol. 20, no. 2, pp. 341-372, 1989.

[2] A. Stern, “Interactive fiction: The story is just beginning,” *IEEE Intelligent Systems & Their Applications,* pp. 16-18, November/December 1998.

[3] J. H. Murray, “Building coherent plots in interactive fiction,” *IEEE Intelligent Systems & Their Applications,* pp. 18-21, November/December 1998.

[4] G. Davenport, “Your Own Virtual Storyworld,” *Scientific American,* vol. 283, no. 5, pp. 79-82, 2000.

[5] J. Pope, “Where do we go from here? Readers' responses to interactive fiction: narrative structures, reading pleasure and the impact of interface design,” *Convergence,* vol. 16, no. 1, pp. 75-94, 2010.

[6] D. Gonzalez and A. S. Gordon, “Comparing speech and text input in interactive narratives,” in *Proceedings of the 23rd International Conference on Intelligent User Interfaces*, March 2018, pp. 141-145.

[7] S. Nesteriuk, “Audiogames: Accessibility and inclusion in digital entertainment,” in *Digital Human Modeling. Applications in Health, Safety, Ergonomics, and Risk Management: 9th International Conference, DHM 2018, Held as Part of HCI International 2018, Las Vegas, NV, USA, July 15-20, 2018, Proceedings 9*, Springer International Publishing, 2018, pp. 338-352.

[8] L. Okkema, “Harvester of Desires: Gaming Amazon Echo through John Cayley’s The Listeners,” in *Proceedings of DiGRA 2018 Conference: The Game is the Message*, January 2018.

[9] N. Röber, C. Huber, K. Hartmann, M. Feustel and M. Masuch, “Interactive audiobooks: combining narratives with game elements,” in *Technologies for Interactive Digital Storytelling and Entertainment: Third International Conference, TIDSE 2006, Darmstadt, Germany, December 4-6, 2006. Proceedings 3*, Springer Berlin Heidelberg, 2006, pp. 358-369.

[10] N. Montfort, “Curveship: An Interactive Fiction System for Interactive Narrating,” in *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, June 2009, pp. 55-62.

[11] D. H. Choi, “LYRA: an Interactive and Interactive Storyteller,” in *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE*

International Conference on Embedded and Ubiquitous Computing (EUC), IEEE, August 2019, pp. 148-153.

[12] B. Swanson and B. Smus, “Usnea: An Authorship Tool for Interactive Fiction using Retrieval Based Semantic Parsing,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, July 2020, pp. 263-269.

[13] M. Hausknecht, P. Ammanabrolu, M. A. Côté and X. Yuan, “Interactive fiction games: A colossal adventure,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, No. 05, April 2020, pp. 7903-7910.

[14] A. S. Fadak and M. O. Khozium, “Dialogue Generation for Digital Interactive Storytelling (Dis),” *International Journal of Future Generation Communication and Networking,* vol. 13, no. 4, pp. 4451-4460, 2020.

[15] IFTF, “Our Mission and Goals,” The Interactive Fiction Technology Foundation, [Online]. Available: https://iftechfoundation.org/mission/. [Accessed 21 November 2024].

[16] IFTF, “Frequently Asked Questions About Interactive Fiction,” The Interactive Fiction Technology Foundation, [Online]. Available: https://iftechfoundation.org/frequently-asked-questions/. [Accessed 21 November 2024].

[17] W. Crowther and D. Woods, “Adventure,” Rick Adams (web version), 1975. [Online]. Available: https://rickadams.org/adventure/. [Accessed 21 November 2024].

[18] IFWiki, “FAQ / What is "interactive fiction"?,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/FAQ. [Accessed 21 November 2024].

[19] ZA/UM, “Disco Elysium,” ZA/UM, 2019.

[20] N. Montfort, “Toward a theory of interactive fiction,” in *IF Theory Reader*, Boston, MA, Transcript On Press, 2011, pp. 25-58.

[21] A. Washenko, “From Infocom to 80 Days: An oral history of text games and interactive fiction,” Ars Technica, 20 June 2024. [Online]. Available: https://arstechnica.com/gaming/2024/06/from-infocom-to-80-days-an-oral-history-of-text-games-and-interactive-fiction/. [Accessed 21 November 2024].

[22] G. Nelson, “A short history of interactive fiction,” in *The Inform Designer's Manual*, 4 ed., The Interactive Fiction Library, 2001.

[23] N. Montfort and E. Short, “Interactive Fiction Communities: From Preservation through Promotion and Beyond,” *Dichtung Digital. Journal für Kunst und Kultur digitaler Medien,* vol. 14, no. 1, pp. 1-14, 2012.

[24] R. S. G. Sorolla, “Crimes Against Mimesis,” in *IF Theory Reader*, Boston, MA, Transcript On Press, 2011, pp. 1-24.

[25] A. Plotkin, “Characterizing, If Not Defining, Interactive Fiction,” in *IF Theory Reader*, Boston, MA, Transcript On Press, 2011, pp. 59-66.

[26] M. Silcox, “not that you may remember time: Interactive Fiction, Stream-of-Consciousness Writing, and Free Will,” in *IF Theory Reader*, Boston, MA, Transcript On Press, 2011, pp. 67-88.

[27] H. Labrande, “Racontons une histoire ensemble: History and Characteristics of French IF,” in *IF Theory Reader*, Boston, MA, Transcript On Press, 2011, pp. 389-432.

[28] F. Cordella, “History of Italian IF,” in *IF Theory Reader*, Boston, MA, Transcript On Press, 2011, pp. 379-388.

[29] D. Adams and S. Meretzky, “Hitchhiker’s Guide to the Galaxy,” Infocom, 1984. [Online]. Available: https://www.bbc.co.uk/programmes/articles/1g84m0sXpnNCv84GpN2PLZG/the-game-30th-anniversary-edition. [Accessed 21 November 2024].

[30] D. Stevens, “10 Years of IF: 1994-2004,” in *IF Theory Reader*, Boston, MA, Transcript On Press, 2011, pp. 359-368.

[31] IFWiki, “Choice-based interactive fiction,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Choice-based_interactive_fiction. [Accessed 21 November 2024].

[32] IFWiki, “Menu-based conversation,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Menu-based_conversation. [Accessed 21 November 2024].

[33] H. Harshitha, “Harshitha, H. "Choose Your Own Adventure: The Evolution of Digital Interactive Fiction and Its Use in Language Pedagogy,” *FORTELL: Journal of Teaching English Language and Literature,* no. 45, pp. 44-55, July 2022.

[34] Torpor Games, “Suzerain,” Torpor Games, Fellow Traveler, 2020.

[35] IFWiki, “Parser-choice hybrid,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Parser-choice_hybrid. [Accessed 21 November 2024].

[36] R. Veeder, “Taco Fiction,” The Interactive Fiction Database, 2011. [Online]. Available: https://ifdb.org/viewgame?id=2ej7ntbmoit9ytvy. [Accessed 21 November 2024].

[37] V. E. Geary, “The Bureau,” In development, [Online]. Available: https://dashingdon.com/play/morbethgames/the-bureau-wip/mygame/. [Accessed 21 November 2024].

[38] inXile Entertainment, “Wasteland 2,” inXile Entertainment, 2014.

[39] S. Barlow, “Her Story,” S. Barlow, 2015.

[40] IFWiki, “Authoring system,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Authoring_system. [Accessed 21 November 2024].

[41] IFWiki, "ADRIFT," The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/ADRIFT. [Accessed 21 November 2024].

[42] C. Wild, “ADRIFT Adventure Development & Runner - Interactive Fiction Toolkit,” [Online]. Available: https://www.adrift.co. [Accessed 21 November 2024].

[43] L. Horsfield, “The Fortress of Fear - The Adventures of Alaric Blackmoon - Episode 4,” ADRIFT Games database, 2013. [Online]. Available: https://www.adrift.co/game/1366. [Accessed 21 November 2024].

[44] IFWiki, “Adventuron,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Adventuron. [Accessed 21 November 2024].

[45] C. Ainsley, “Adventuron Classroom,” Adventuron Software Limited, UK, [Online]. Available: https://adventuron.io. [Accessed 21 November 2024].

[46] M. Sagra, “The Mansion,” Manuel Sagra, 2019. [Online]. Available: https://manuelsagra.itch.io/the-mansion. [Accessed 21 November 2024].

[47] IFWiki, “Dialog,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Dialog. [Accessed 21 November 2024].

[48] L. Åkesson, “Dialog,” [Online]. Available: https://linusakesson.net/dialog/. [Accessed 21 November 2024].

[49] L. Åkesson, “Dialog,” L. Åkesson, 2021.

[50] Borogove, “The Borogove IDE,” The Interactive Fiction Technology Foundation, [Online]. Available: https://borogove.app/. [Accessed 21 November 2024].

[51] IFWiki, “Inform,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Inform. [Accessed 21 November 2024].

[52] IFWiki, “Inform 6,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Inform_6. [Accessed 21 November 2024].

[53] IFWiki, “Inform 7,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Inform_7. [Accessed 21 November 2024].

[54] G. Nelson, “Inform 7,” [Online]. Available: https://github.com/ganelson/inform. [Accessed 21 November 2024].

[55] G. Nelson, “Inform,” G. Nelson, 2022.

[56] IFWiki, “TADS 3,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/TADS_3. [Accessed 21 November 2024].

[57] M. J. Roberts, “TADS - The Text Adventure Development System,” [Online]. Available: http://www.tads.org. [Accessed 21 November 2024].

[58] M. J. Roberts, “Return to Ditch Day,” The Interactive Fiction Database, 2004. [Online]. Available: https://ifdb.org/viewgame?id=sicva377zqygxcq2. [Accessed 21 November 2024].

[59] IFWiki, “ZIL,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/ZIL. [Accessed 21 November 2024].

[60] IFWiki, “ChoiceScript,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/ChoiceScript. [Accessed 21 November 2024].

[61] Choice of Games LLC, “Introduction to ChoiceScript,” [Online]. Available: https://www.choiceofgames.com/make-your-own-games/choicescript-intro/. [Accessed 21 November 2024].

[62] D. Fabulich, “ChoiceScript,” Choice of Games, [Online]. Available: https://github.com/dfabulich/choicescript. [Accessed 21 November 2024].

[63] J. S. Hill, “Choice of the Vampire,” Choice of Games, 2010.

[64] aucchen (GitHub username), “The Dendry GitHub Repository,” [Online]. Available: https://github.com/aucchen/dendry. [Accessed 21 November 2024].

[65] J. Zamor, “Getting started with Dendry,” [Online]. Available: https://smwhr.notion.site/Getting-started-with-Dendry-188e7e39a961497fb2d0a0deee0c21a0. [Accessed 21 November 2024].

[66] J. Zamor, “Treasure Island,” Dendry Tutorial, [Online]. Available: https://smwhr.github.io/dendry-tutorial/. [Accessed 21 November 2024].

[67] inkle Ltd., “ink: A narrative scripting language for games,” [Online]. Available: https://www.inklestudios.com/ink/. [Accessed 21 November 2024].

[68] P. Joannon, “Ink integration for Godot engine,” [Online]. Available: https://github.com/paulloz/godot-ink. [Accessed 21 November 2024].

[69] JBenda (GitHub username), “Inkle Ink C++ runtime with JSON>Binary compiler,” [Online]. Available: https://github.com/JBenda/inkcpp. [Accessed 21 November 2024].

[70] R. Silin, “The Ink language parser and runtime implementation in Lua,” [Online]. Available: https://github.com/astrochili/narrator. [Accessed 21 November 2024].

[71] Y. Lohse, ”A JavaScript port of inkle's ink scripting language,” [Online]. Available: https://github.com/y-lohse/inkjs. [Haettu 21 November 2024].

[72] F. Maquin, ”Implementation of inkle's Ink in pure GDScript for Godot,” [Online]. Available: https://github.com/ephread/inkgd. [Haettu 21 November 2024].

[73] inkle Ltd., Cape Guy Ltd., “80 Days,” inkle Ltd., 2015.

[74] IFWiki, “Twine,” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Twine. [Accessed 21 November 2024].

[75] IFTF, “Twine website,” The Interactive Fiction Technology Foundation, [Online]. Available: https://twinery.org. [Accessed 21 November 2024].

[76] T. M. Edwards, “Tweego command line compiler for Twine/Twee,” 2014-2024. [Online]. Available: https://github.com/tmedwards/tweego. [Accessed 21 November 2024].

[77] C. Klimas, “Twine,” The Interactive Fiction Technology Foundation, 2024.

[78] J. Zamor, “binksi,” [Online]. Available: https://smwhr.itch.io/binksi. [Accessed 21 November 2024].

[79] J. Zamor, “binksi repository,” [Online]. Available: https://www.github.com/smwhr/binksi. [Accessed 21 November 2024].

[80] Ti R² (itch.io username) & Zamor, J., “Is there a way to use this together with Bitsy 3D?,” binksi community forum, 14 April 2024. [Online]. Available: https://itch.io/t/3670539/is-there-a-way-to-use-this-together-with-bitsy-3d. [Accessed 21 November 2024].

[81] A. Le Doux, “bitsy,” [Online]. Available: https://www.bitsy.org. [Accessed 21 November 2024].

[82] A. Le Doux, “bitsy editor,” [Online]. Available: https://make.bitsy.org/. [Accessed 21 November 2024].

[83] IFWiki, “Quest (Language),” The Interactive Fiction Wiki, [Online]. Available: https://www.ifwiki.org/Quest_(Language). [Accessed 21 November 2024].

[84] textadventures.co.uk, “Quest,” [Online]. Available: https://textadventures.co.uk/quest. [Accessed 21 November 2024].

[85] A. Joel, “Quest,” textadventures.co.uk, 2017/2018.

[86] Ren'Py, “Ren'Py website,” [Online]. Available: https://www.renpy.org. [Accessed 21 November 2024].

[87] Ren'Py, “The Ren'Py Visual Novel Engine,” Ren'Py, 2024.

[88] Microsoft, “Azure AI Speech,” Microsoft Azure, [Online]. Available: https://azure.microsoft.com/en-us/products/ai-services/ai-speech. [Accessed 21 November 2024].

[89] Microsoft, “Speech Technologies,” Microsoft Learn, [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/office/developer/speech-technologies/. [Accessed 21 November 2024].

[90] Microsoft, “Quickstart: Convert text to speech,” Microsoft Learn, [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/speech-service/get-started-text-to-speech. [Accessed 21 November 2024].

[91] T. V. Raman, “Emacspeak -- The Complete Audio Desktop,” Emacspeak Inc, [Online]. Available: https://emacspeak.sourceforge.net. [Accessed 21 November 2024].

[92] eSpeak NG, “eSpeak NG speech syntesizer,” 2022. [Online]. Available: https://github.com/espeak-ng/espeak-ng/. [Accessed 21 November 2024].

[93] Freedom Scientific, “JAWS Headquarters,” [Online]. Available: https://support.freedomscientific.com/JAWSHQ/JAWSHeadquarters01. [Accessed 21 November 2024].

[94] Microsoft, “Complete Guide to Narrator,” Microsoft Support, [Online]. Available: https://support.microsoft.com/en-us/windows/complete-guide-to-narrator-e4397a0d-ef4f-b386-d8ae-c172f109bdb1. [Accessed 21 November 2024].

[95] Naturalsoft Ltd., ”NaturalReader,” [Online]. Available: https://www.naturalreaders.com. [Haettu 21 November 2024].

[96] NV Access, ”About NVDA,” [Online]. Available: https://www.nvaccess.org/about-nvda/. [Haettu 21 November 2024].

[97] Google, “TalkBack,” Android Support, [Online]. Available: https://support.google.com/accessibility/android/topic/3529932. [Accessed 21 November 2024].

[98] Apple, “Vision,” Apple Accessibility, [Online]. Available: https://www.apple.com/accessibility/vision/. [Accessed 21 November 2024].

[99] Amazon, “Alexa Help And Resources,” [Online]. Available: https://www.amazon.com/alexa-help-resources/b?ie=UTF8&node=21576560011. [Accessed 21 November 2024].

[100] Amazon, “Alexa Skills Kit,” Amazon Developer, [Online]. Available: https://developer.amazon.com/en-US/alexa/alexa-skills-kit. [Accessed 21 November 2024].

[101] Amazon, “About Alexa for Apps,” Amazon Developer Documentation, [Online]. Available: https://developer.amazon.com/en-US/docs/alexa/alexa-for-apps/about-alexa-for-apps.html. [Accessed 21 November 2024].

[102] AlphaCephei, “CMUSphinx Open Source Speehc Recognition Toolkit,” [Online]. Available: https://cmusphinx.github.io. [Accessed 21 November 2024].

[103] Nuance Communications Inc., “Dragon Speech Recognition Solutions,” [Online]. Available: https://www.nuance.com/dragon.html. [Accessed 21 November 2024].

[104] Google, “Google Assistant,” [Online]. Available: https://assistant.google.com/learn/. [Accessed 21 November 2024].

[105] Google, “Google Assistant for Developers,” Google Developers, [Online]. Available: https://developers.google.com/assistant. [Accessed 21 November 2024].

[106] HTK Team, “HTK Speech Recognition Toolkit,” Cambridge University Engineering Department, [Online]. Available: https://htk.eng.cam.ac.uk. [Accessed 21 November 2024].

[107] Startseite CMU-KIT, “Janus Recognition Toolkit,” Interactive Systems Labs at Carnegie Mellon University and Karlsruhe Institute of Technology, [Online]. Available: https://isl.anthropomatik.kit.edu/english/3498.php. [Accessed 21 November 2024].

[108] A. Lee, “Julius: Open-Source Large Vocabulary Continuous Speech Recognition Engine,” Kawahara Lab., Kyoto University, Julius project team, Lee Lab., Nagoya Institute of Technology, [Online]. Available: https://github.com/julius-speech/julius. [Accessed 21 November 2024].

[109] Povey, Daniel and Ghoshal, Arnab and Boulianne, Gilles and Burget, Lukas and Glembek, Ondrej and Goel, Nagendra and Hannemann, Mirko and Motlicek, Petr and Qian, Yanmin and Schwarz, Petr and Silovsky, Jan and Stemmer, Georg and Vesely, Karel, “The Kaldi Speech Recognition Toolkit,” [Online]. Available: http://kaldi-asr.org/doc/. [Accessed 21 November 2024].

[110] Microsoft, “End of support for Cortana,” Microsoft Support, [Online]. Available: https://support.microsoft.com/fi-fi/topic/end-of-support-for-cortana-d025b39f-ee5b-4836-a954-0ab646ee1efa. [Accessed 21 November 2044].

[111] Microsoft, “Use voice access to control your PC & author text with your voice,” Microsoft Support, [Online]. Available: https://support.microsoft.com/en-us/topic/use-voice-access-to-control-your-pc-author-text-with-your-voice-4dcd23ee-f1b9-4fd1-bacc-862ab611f55d. [Accessed 21 November 2024].

[112] Apple, “Apple Siri,” [Online]. Available: https://www.apple.com/siri/. [Accessed 21 November 2024].

[113] Apple, “Sirikit,” Apple Developer, [Online]. Available: https://developer.apple.com/documentation/sirikit. [Accessed 21 November 2024].

[114] Apple, ”Speech,” Apple Developer, [Online]. Available: https://developer.apple.com/documentation/speech/. [Haettu 21 November 2024].

[115] OpenAI, ”Introducing Whisper,” OpenAI, September 2022. [Online]. Available: https://openai.com/index/whisper/. [Haettu 21 November 2024].

[116] Alpha Cephei, “Vosk API repository,” [Online]. Available: https://github.com/alphacep/vosk-api. [Accessed 21 November 2024].

[117] The Apache Software Foundation, “Apache License, Version 2.0,” [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0. [Accessed 21 November 2024].

[118] Alpha Cephei, “Vosk Speech Recognition Toolkit manual,” [Online]. Available: https://alphacephei.com/vosk/. [Accessed 21 November 2024].

[119] jonsd, mavison, rhdunn, valdisvi (SourceForge usernames), “eSpeak: speech synthesis,” 2021. [Online]. Available: https://espeak.sourceforge.net. [Accessed 21 November 2024].

[120] D. G. Jerz, “Wander (1974) — a lost mainframe game is found!,” 22 April 2015. [Online]. Available: https://jerz.setonhill.edu/blog/2015/04/22/wander-1974-a-lost-mainframe-game-is-found/. [Accessed 21 November 2024].

Appendices

Appendix A The questionnaire

These are the questions in the questionnaire. Questions marked with * were mandatory.

1. Have you played interactive fiction before?*
    - Yes, plenty
    - Yes, some
    - No, but I'm aware of the concept
    - No, it is completely unfamiliar
2. Would you be interested in playing voice-based interactive fiction again?*
    - Yes
    - Maybe, but it would need to be improved
    - No, I didn't like the voice-based experience
    - No, I don't like interactive fiction

General questions about the experience

3. What is your first language?*

This is needed in order to see if the reliability of speech recognition in the game is affected by accent.

- Finnish
- Swedish
- English
- Something else (what?)

4. Did you complete the game?*

    - Yes
    - No, it stopped responding
    - No, it was unplayable

Specific questions about the experience

5. Did you have trouble following the story?*

    - Never 1 … 2 … 3 … 4 … 5 Always

6. Did you have trouble following the speech?*

    - Never 1 … 2 … 3 … 4 … 5 Always

7. Did you use the “repeat options” command?*

    - Never 1 … 2 … 3 … 4 … 5 Always

8. Did you need to repeat the commands in order to get the game to recognize and accept them?*

    - Never 1 … 2 … 3 … 4 … 5 Always

Did you ned to repeat the commands more than once for any given command?*

For example, say “stop” for five times before the game closes

- Never 1 … 2 … 3 … 4 … 5 Always

Improvement questions

Answering these questions is voluntary

9. What was good in this system?

10. What in this system needs improving?